

Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

А. С. Кобайло, Н. А. Жиляк

ВВЕДЕНИЕ В XML

*Рекомендовано
учебно-методическим объединением
учреждений высшего образования Республики Беларусь
по образованию в области информатики и радиоэлектроники
в качестве учебно-методического пособия для студентов
учреждений высшего образования по направлению
специальности «Информационные системы и технологии
(издательско-полиграфический комплекс)»*

Минск 2011

УДК 004.434(075.8)
ББК 32.97я73
К55

Р е ц е н з е н т ы :

кафедра информатики Белорусского государственного
университета информатики и радиоэлектроники
(доктор физико-математических наук, профессор,
заведующий кафедрой *Л. И. Минченко*);

кандидат физико-математических наук,
доцент кафедры программного обеспечения телекоммуникаций
Высшего государственного колледжа связи *В. Ф. Бондаренко*

Все права на данное издание защищены. Воспроизведение всей книги или ее части не может быть осуществлено без разрешения учреждения образования «Белорусский государственный технологический университет».

Кобайло, А. С.

К55 Введение в XML : учеб.-метод. пособие для студентов по направлению специальности «Информационные системы и технологии (издательско-полиграфический комплекс)» / А. С. Кобайло, Н. А. Жияк. – Минск : БГТУ, 2011. – 321 с.
ISBN 978-985-530-101-2.

Учебно-методическое пособие содержит основные понятия, принципы организации, описание возможностей расширяемого языка разметки Extensible Markup Language (XML). По всем основным темам предлагаются задания для самостоятельного выполнения.

Издание предназначено для студентов, изучающих дисциплину «Информационные технологии», а также может быть полезно студентам инженерно-экономического факультета при изучении дисциплины «Компьютерные информационные технологии».

**УДК 004.434(075.8)
ББК 32.97я73**

ISBN 978-985-530-101-2

© Кобайло А. С., Жияк Н. А., 2011
© УО «Белорусский государственный
технологический университет», 2011

ОГЛАВЛЕНИЕ

Введение	9
Глава 1. ПЕРВЫЕ ШАГИ В XML	11
1.1. Знакомство с XML	11
1.2. Предназначение XML	11
1.3. Язык XML решает проблемы	15
1.4. Создание XML-документов	17
1.5. Отображение XML-документов	18
1.6. Таблица стилей	18
1.7. Связывание данных	18
1.8. Написание сценария	19
1.9. SGML, HTML и XML	19
1.10. Заменит ли XML HTML?	20
1.11. Официальные концептуальные цели XML	21
1.12. Стандартные XML-приложения	23
1.13. Реальное использование XML	24
Задания	27
Глава 2. РАБОТА С XML-ДОКУМЕНТАМИ	28
2.1. Создание и отображение вашего первого XML-документа	28
2.1.1. Создание XML-документа	28
2.1.2. Пролог	28
2.1.3. Элемент Документ	29
2.2. Некоторые базовые правила XML	31
2.3. Отображение XML-документа	32
2.4. Обнаружение ошибок XML в Internet Explorer 5	32
Задания	33
Глава 3. ЭЛЕМЕНТЫ И АТТРИБУТЫ XML-ДОКУМЕНТОВ.....	41
3.1. Создание корректно сформированных XML-документов	41
3.1.1. Составные части корректно сформированного XML-документа	41
3.1.2. Наименьший XML-документ	42
3.1.3. Добавление элементов в документ	43
.....	3

3.2. Анатомия элемента	46
3.2.1. Типы содержимого элемента	48
3.2.2. Пустые элементы	50
3.3. Задание атрибутов для элементов	51
3.3.1. Правила для создания атрибутов	52
3.3.2. Правила для корректного задания значений атрибутов	53
Задания	54
 Глава 4. КОММЕНТАРИИ В XML. РАЗДЕЛЫ CDATA	57
4.1. Добавление комментариев, инструкций по обработке и разделов CDATA	57
4.2. Добавление комментариев	57
4.2.1. Форма записи комментариев	58
4.2.2. Место для размещения комментария	58
4.3. Добавление инструкций по обработке	59
4.3.1. Форма записи инструкций по обработке	59
4.3.2. Особенности использования инструкций по обработке	60
4.3.3. Место для размещения инструкций по обработке	60
4.4. Добавление разделов CDATA	62
4.4.1. Форма записи раздела CDATA	62
4.4.2. Место для размещения раздела CDATA	63
 Глава 5. ВАЛИДНОСТЬ XML-ДОКУМЕНТОВ. ПОДМНОЖЕСТВА DTD	65
5.1. Создание валидных XML-документов	65
5.1.1. Основной критерий для валидного документа	65
5.1.2. Требования корректности формирования и валидности	65
5.1.3. Преимущества использования валидных XML-документов	66
5.2. Форма записи DTD	67
5.2.1. Создание DTD	68
5.2.2. Добавление DTD	69
5.3. Объявление типов элементов	70
5.4. Описание содержимого элемента	72
5.4.1. Задание содержимого элемента	73
5.4.2. Модель содержимого	74

5.5. Задание смешанного содержимого	78
5.6. Объявление атрибутов	80
5.6.1. Форма записи объявления списка атрибутов	80
5.6.2. Тип атрибута	82
5.7. Задание маркерного типа	83
5.8. Задание нумерованных типов	87
5.9. Объявление значения по умолчанию	89
5.10. Использование внешних подмножеств DTD	91
5.10.1. Использование только внешнего подмножества DTD	91
5.10.2. Использование и внешних, и внутренних подмножеств DTD	92
5.10.3. Условия игнорирования разделов внешнего подмножества DTD	94
Задания	95
 Глава 6. ПРИМИТИВЫ	100
6.1. Определение и использование примитивов	100
6.1.1. Определения и классификация примитивов	100
6.1.2. Внешнее подмножество DTD	100
6.1.3. Типы примитивов	102
6.2. Объявление общих примитивов	104
6.2.1. Объявление общего внутреннего разбираемого примитива	104
6.2.2. Объявление общего внешнего разбираемого примитива	106
6.2.3. Объявление общего внешнего неразбираемого примитива	107
6.3. Объявление нотаций	109
6.4. Объявление параметрических примитивов	110
6.4.1. Объявление параметрического внутреннего разбираемого примитива	110
6.4.2. Объявление параметрического внешнего разбираемого примитива	112
6.5. Вставка ссылок на примитив	115
6.6. Вставка ссылок на символы	117
6.7. Использование предварительно определенных примитивов	119
6.8. Объявление документа автономным (standalone)	120
Задания	120

Глава 7. ТАБЛИЦА СТИЛЕЙ	125
7.1. Отображение XML-документов с использованием таблиц каскадных стилей	125
7.2. Основные этапы при использовании таблицы каскадных стилей	126
7.2.1. Нечувствительность к регистру в CSS	131
7.2.2. Наследование установок свойств	132
7.2.3. Использование множественных элементов и множественных правил	133
7.2.4. Использование контекстуальных селекторов	134
7.2.5. Использование атрибута STYLE	135
7.2.6. Импорт других таблиц стилей	136
7.2.7. Задание значений URL	137
7.2.8. Присвоение значений в таблицах каскадных стилей	140
7.2.9. Установка свойства display	143
7.2.10. Задание ключевых слов CSS в качестве значений	146
7.3. Установка свойств шрифта	146
7.3.1. Установка свойства font-family	146
7.3.2. Установка свойства font-size	149
7.3.3. Задание значений в процентах	151
7.3.4. Задание значений в размерных единицах	151
7.3.5. Установка свойства font-style	153
7.3.6. Установка свойства font-weight	154
7.3.7. Установка свойства font-variant	154
7.4. Установка свойства color	155
7.5. Установка свойств фона	157
7.5.1. Установка свойства background-color	158
7.5.2. Установка свойства background-image	158
7.5.3. Установка свойства background-repeat	160
7.5.4. Установка свойства background-position	163
7.6. Установка свойств разбивки текста и выравнивания	168
7.6.1. Установка свойства letter-spacing	168
7.6.2. Установка свойства vertical-align	169
7.6.3. Установка свойства text-align	170
7.6.4. Установка свойства text-indent	172
7.6.5. Установка свойства line-height	174
7.6.6. Установка свойства text-transform	175
7.6.7. Установка свойства text-decoration	175

7.7. Установка свойств текстовых областей	176
7.8. Установка свойств управления полями	177
7.9. Установка свойств управления обрамлением	180
7.9.1. Установка свойства border-style	180
7.9.2. Установка свойства border-width	181
7.9.3. Установка свойства border-color	182
7.10. Установка свойств просвета между обрамлением и текстом	184
7.11. Установка свойств размеров	185
7.12. Установка свойств позиционирования	187
7.12.1. Установка свойства float	187
7.12.2. Установка свойства clear	188
7.13. Вставка элементов HTML в XML-документы и использование пространства имен	190
7.14. Создание и использование полноценной таблицы стилей	195
Задания	195

Глава 8. ОТОБРАЖЕНИЕ XML-ДОКУМЕНТА.

СВЯЗЬ ДАННЫХ	205
8.1. Основные шаги	205
8.2. Как хранятся данные XML	207
8.3. Проверка на наличие ошибок XML	208
8.4. Использование табличного сцепления данных	209
8.4.1. Использование одной HTML-таблицы для отображения простого набора записей	209
8.4.2. Использование постраничного отображения	213
8.4.3. Использование вложенных таблиц для отображения иерархической структуры записей	219
8.5. Использование связывания данных по одной записи	225
8.5.1. Перемещение между записями	227
8.5.2. Другие способы связывания данных	230
8.5.3. Связывание с другими HTML-элементами	232
8.6. Передача HTML-разметки	235
8.7. Обновление накопленных данных XML	237
8.8. Использование DTD при связывании данных	238
8.9. Связывание HTML-элементов с XML-атрибутами	243
8.10. Использование сценариев для DSO	248
Задания	253

Глава 9. СТРУКТУРА УЗЛОВ XML-ДОКУМЕНТОВ	255
9.1. Связывание XML-документа с HTML-страницей	255
9.2. Структура DOM	256
9.3. Доступ и отображение элементов XML-документа	262
9.4. Использование объекта NodeList	267
9.5. Извлечение символьных данных элемента	268
9.6. Отображение переменного числа XML-элементов	270
9.7. Использование других способов доступа к элементам	273
9.8. Доступ и отображение значений атрибутов в XML-документе	277
9.9. Доступ к примитивам и нотациям XML	279
9.10. Перемещение внутри XML-документа	284
9.11. Работа страницы проверки на валидность	288
Задания	291
Глава 10. ФИЛЬТРАЦИЯ И СОРТИРОВКА ДАННЫХ XML	294
10.1. Основы использования XSL-таблиц стилей	294
10.2. Использование одного шаблона XSL	295
10.3. Отображение переменного числа элементов	301
10.4. Использование нескольких шаблонов	309
10.5. Фильтрация и сортировка данных XML	311
10.5.1. Фильтрация	312
10.5.2. Сортировка	313
10.5.3. Пример таблицы стилей, осуществляющей фильтрацию и сортировку	313
10.6. Доступ к атрибутам XML	316
Задания	319
Литература.....	320

ВВЕДЕНИЕ

XML (Extensible Markup Language) – это язык разметки, описывающий целый класс объектов данных, называемых XML-документами. Этот язык используется в качестве средства для описания грамматики других языков и контроля за правильностью составления документов. Другими словами, сам по себе XML не содержит никаких тегов, предназначенных для разметки, он просто определяет порядок их создания.

Процесс создания XML-документа очень прост и требует от нас лишь базовых знаний HTML и понимания тех задач, которые мы хотим выполнить, используя XML в качестве языка разметки. Таким образом, у программистов появляется уникальная возможность разрабатывать собственные команды, позволяющие им наиболее эффективно определять данные, содержащиеся в документе. Автор документа создает его структуру, строит необходимые связи между элементами, используя те команды, которые удовлетворяют его требованиям, и добивается такого типа разметки, который необходим ему для выполнения операций просмотра, поиска, анализа документа.

Еще одним из очевидных достоинств XML является возможность использования его в качестве универсального языка запросов к хранилищам информации. Сегодня в консорциуме W3C находится на рассмотрении рабочий вариант стандарта XML-QL (или XQL), который, возможно, в будущем составит серьезную конкуренцию SQL. Кроме того, XML-документы могут выступать в качестве уникального способа хранения данных, который включает в себя одновременно средства для разбора информации и представления ее на стороне клиента. В этой области одним из перспективных направлений является интеграция Java и XML-технологий, позволяющая использовать мощь обеих технологий при построении машинно-независимых приложений, в основе которых лежит универсальный формат данных при обмене информацией.

XML позволяет также осуществлять контроль за корректностью данных, хранящихся в документах, производить проверки иерархических соотношений внутри документа и устанавливать единый стандарт на структуру документов, содержимым которых могут быть самые различные данные. Это означает, что его можно использовать

при построении сложных информационных систем, в которых очень важным является вопрос обмена информацией между различными приложениями, работающими в одной системе. Создавая структуру механизма обмена информацией в самом начале работы над проектом, менеджер может избавить себя в будущем от многих проблем, связанных с несовместимостью используемых различными компонентами системы форматов данных.

Также одним из достоинств XML является то, что программы-обработчики XML-документов не сложны, и уже сегодня появились и свободно распространяются всевозможные программные продукты, предназначенные для работы с XML-документами. XML поддерживается сегодня в Microsoft Internet Explorer 4 и в бэта-версиях IE5. Было заявлено о его поддержке в последующих версиях Netscape Communicator, СУБД Oracle, DB-2, в приложениях Microsoft Office. Все это дает основания предполагать, что, скорее всего, в ближайшем будущем XML станет основным языком обмена информацией для информационных систем, заменив собой, тем самым, HTML. На основе XML уже сегодня созданы такие известные специализированные языки разметки, как SMIL, CDF, MathML, XSL, и список рабочих проектов новых языков, находящихся на рассмотрении W3C, постоянно пополняется.

Глава 1. ПЕРВЫЕ ШАГИ В XML

1.1. ЗНАКОМСТВО С XML

Язык XML (Extensible Markup Language) был разработан рабочей группой XML Working Group консорциума World Wide Web Consortium (W3C). Вот как описывают его создатели: «Расширяемый язык разметки Extensible Markup Language (XML) представляет собой составную часть языка SGML. Он предназначен для облегчения использования языка SGML в Web и выполнения задач, которые в настоящее время реализуются с помощью языка HTML. XML разработан с целью усовершенствовать применение и взаимодействие языков SGML и HTML».

XML – язык разметки, разработанный специально для размещения информации в World Wide Web, аналогично языку гипертекстовой разметки HTML (Hypertext Markup Language), который изначально стал стандартным языком создания Web-страниц. Поскольку язык HTML полностью удовлетворяет всем нашим потребностям, возникает вопрос: для чего понадобился совершенно новый язык для Web? В чем состоят его преимущества и достоинства? Как он взаимодействует с HTML? Заменит ли он HTML, или только усовершенствует его? Наконец, что собой представляет язык SGML, частью которого является XML, и почему нельзя использовать для Web-страниц собственно SGML? В этой главе мы постараемся ответить на все эти вопросы.

1.2. ПРЕДНАЗНАЧЕНИЕ XML

Язык HTML предоставляет фиксированный набор элементов, которые вы можете использовать для размещения компонентов на типовой Web-странице. Примерами таких элементов являются заголовки, абзацы, списки, таблицы, изображения и связи. Например, HTML отлично подходит для создания личной домашней страницы. Ниже приведено описание домашней страницы в кодах HTML.

Листинг 1.1. Описание домашней страницы

```
<HTML>
<HEAD>
<TITLE>Home Page</TITLE>
```

```

</HEAD>
<BODY>
<H1><IMG SRC="MainLogo.gif"> Michael Young's Home Page</H1>
<P><EM>Welcome to my Web site!</EM></P>
<H2>Web Site Contents</H2>
<P>Please choose one of the following topics:</P>
<UL>
<LI><A Href="Writing.htm"><B>Writing</B></A></LI>
<LI><A Href="Family.htm"><B>Family</B></A></LI>
<LI><A Href="Photos.htm"><B>Photo Gallery</B></A></LI>
</UL>
<H2>Other Interesting Web Sites</H2>
<P>Click one of the following to explore another Web site:</P>
<UL>
<LI><A HREF=http://www.yahoo.com/>Yahoo Search Engine</A></LI>
<LI><A HREF=http://www.amazon.com/>Amazon Bookstore</A></LI>
<LI><A HREF=http://mspress.microsoft.com/>Microsoft Press</A></LI>
</UL>
</BODY>
</HTML>

```

В Microsoft Internet Explorer 5 эта страница будет выглядеть так, как показано на рис. 1.1.

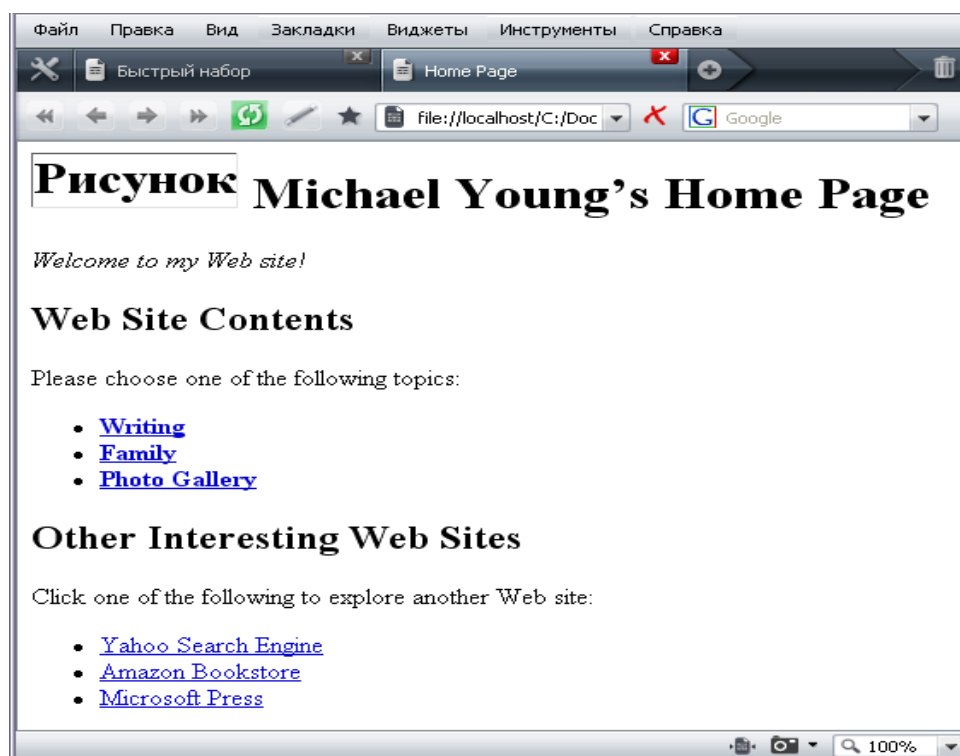


Рис. 1.1. Отображение в Internet Explorer 5 домашней страницы

Каждый элемент начинается с начального тега (текста, заключенного в угловые скобки (< >)), который содержит имя элемента и дополнительную информацию. Большинство элементов заканчиваются конечным тегом, который повторяет соответствующий начальный тег, за исключением того, что имеет символ косой черты (/) перед именем элемента. Элементарное содержание представляет собой текст, который расположен между начальным и конечным тегами (рис. 1.2). Обратите внимание, что многие элементы в предыдущем примере содержат вложенные элементы.



Рис. 1.2. Элементарное содержание элемента

В рассматриваемом примере HTML-страницы имеются следующие элементы, представленные в таблице.

Элементы HTML-страницы

Элемент HTML	Составляющая разметки страницы
HTML	Вся страница
HEAD	Информация о заголовке, например название страницы
TITLE	Название страницы, которое появляется в строке заголовка окна браузера
BODY	Основной текст, отображаемый браузером
H1	Заголовок верхнего уровня
H2	Заголовок второго уровня
P	Абзац текста
UL	Маркированный список (Unordered List)
LI	Отдельный элемент в списке (List Item)
IMG	Изображение
A	Связь с другой страницей или с другим местом данной страницы (элемент Anchor)
EM	Блок текста, набранного курсивом (EMphasized)
B	Блок текста, набранного полужирным шрифтом

Браузер, отображающий HTML-страницу, распознает каждый из этих стандартных элементов и показывает их в соответствующем формате. Например, обычно браузер отображает заголовок H1 наибольшим размером шрифта, заголовок H2 – меньшим размером шрифта, а элемент P – еще меньшим размером шрифта. Элемент LI отображается как абзац текста в составе маркированного списка. Элемент A браузер преобразует в ссылку (подчеркнутый текст), на которой пользователь может щелкнуть, чтобы перейти в другое место текущей страницы или на другую страницу. Хотя набор HTML-элементов был существенно расширен по сравнению с первой версией HTML, язык HTML по-прежнему не пригоден для представления многих типов документов. Ниже приведены примеры документов, которые не могут быть адекватно описаны с помощью языка HTML:

1) документ, который не содержит типовых компонентов (заголовков, абзацев, списков, таблиц и т. д.). Например, в языке HTML отсутствуют элементы, необходимые для отображения музыкальных символов или математических уравнений;

2) база данных, такая как каталог книг. Вы можете использовать HTML-страницу, чтобы хранить и отображать информацию из статической базы данных (например, перечень книг и их описание). Однако, если вам понадобится осуществить сортировку, фильтрацию, поиск и обработку информации, придется снабдить каждую из составных частей информации соответствующей меткой (как в программе, работающей с базами данных, такой как Microsoft Access). В языке HTML не предусмотрено соответствующих элементов;

3) документ, который вы хотите представить в виде иерархической структуры. Допустим, вы пишете книгу и хотите разбить ее на части, главы, разделы A, B, C и т. д. В дальнейшем программа может использовать данную структуру документа для создания оглавления, оформления различных уровней в структуре с помощью различных стилей, извлечения определенных разделов, а также обработки информации иными способами. Однако элемент типа заголовок в HTML содержит лишь описание собственно текста, например содержимое Web-сайта.

Поскольку внутри элемента типа заголовок вы не задаете вложенные элементы текста, которые относятся к разделам документа, эти элементы не могут быть использованы для представления иерархической структуры документа.

Язык XML позволяет преодолеть эти ограничения.

1.3. ЯЗЫК XML РЕШАЕТ ПРОБЛЕМЫ

Описание на языке XML представляет собой операторы, написанные с соблюдением определенного синтаксиса. Когда вы создаете XML-документ, то вместо использования ограниченного набора определенных элементов вы имеете возможность создавать ваши собственные элементы и присваивать им любые имена по вашему выбору – именно поэтому язык XML является расширяемым (extensible). Следовательно, вы можете использовать XML для описания практически любого документа, от музыкальной партитуры до базы данных. Например, вы можете описать перечень книг, подобно представленному в следующем XML-документе.

Листинг 1.2. Перечень книг

```
<?xml version="1.0"?><INVENTORY>
  <BOOK><TITLE>The Adventures of Huckleberry Finn</TITLE>
    <AUTHOR>Mark Twain</AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>298</PAGES>
    <PRICE>$5.49</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>Moby-Dick</TITLE>
    <AUTHOR>Herman Melville</AUTHOR>
    <BINDING>trade paperback</BINDING>
    <PAGES>605</PAGES>
    <PRICE>$4.95</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>The Scarlet Letter</TITLE>
    <AUTHOR>Nathaniel Hawthorne</AUTHOR>
    <BINDING>trade paperback</BINDING>
    <PAGES>253</PAGES>
    <PRICE>$4.25</PRICE>
  </BOOK>
</INVENTORY>
```

В Microsoft Internet Explorer 5 эта страница будет отображена, как показано на рис. 1.3.



Примечание. Для описания базы данных в XML предусмотрена возможность работы с несколькими форматами (например, формат .mdb Access или .dbf dBase): язык XML построен на принципе открытых и доступных стандартов.

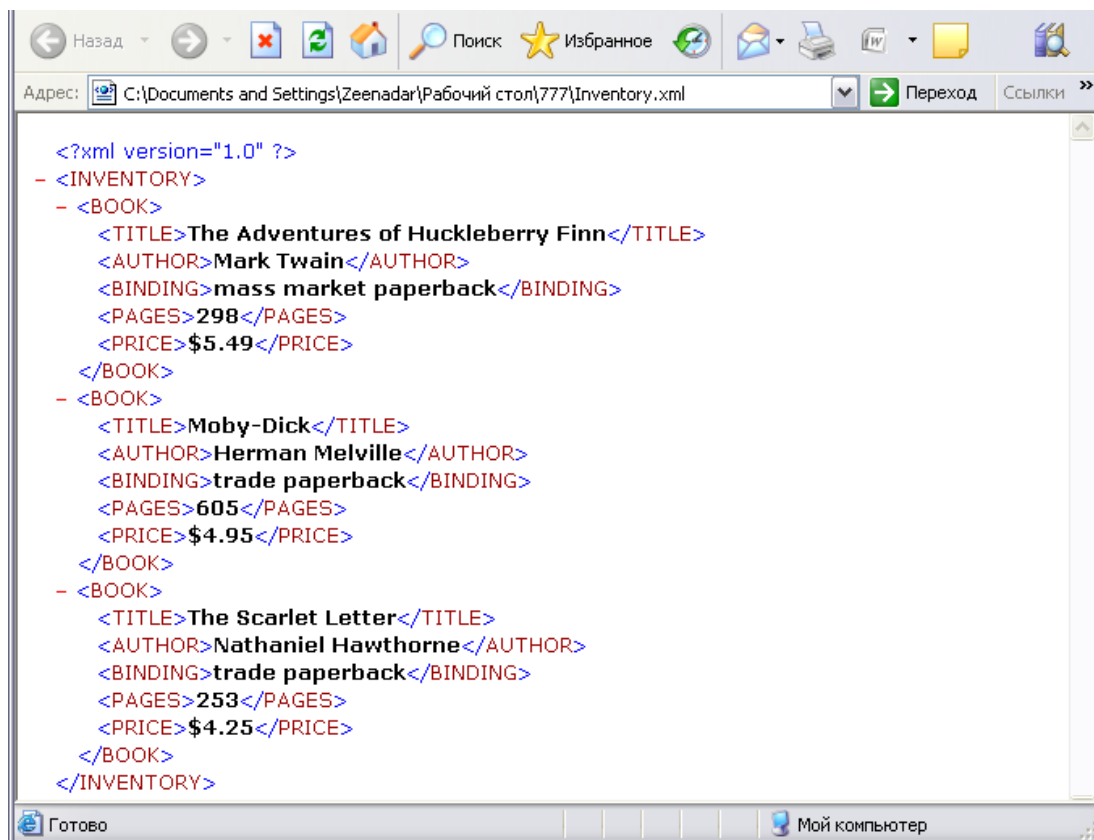


Рис. 1.3. Отображение в Internet Explorer 5 страницы «Перечень книг»

Имена элементов в XML-документе (такие как INVENTORY, BOOK и TITLE в приведенном выше примере) не являются определениями языка XML. Вы всего лишь назначаете эти имена при создании определенного документа. Для ваших элементов вы можете выбирать любые корректно заданные имена (LIST вместо INVENTORY, либо ITEM вместо BOOK).



Совет. Когда вы присваиваете имена в XML-документе, старайтесь делать их по возможности наиболее информативными. Одним из преимуществ XML-документа является то, что каждому фрагменту информации может быть присвоено информативное описание.

В предыдущем примере XML-документ имеет иерархическую структуру в виде дерева с элементами, вложенными в другие элементы, и с одним элементом верхнего уровня (в нашем примере – INVENTORY) – он носит название элемент Документ, или корневой элемент, – который содержит все другие элементы. Структуру описанного в примере документа можно представить, как показано на рис. 1.4.

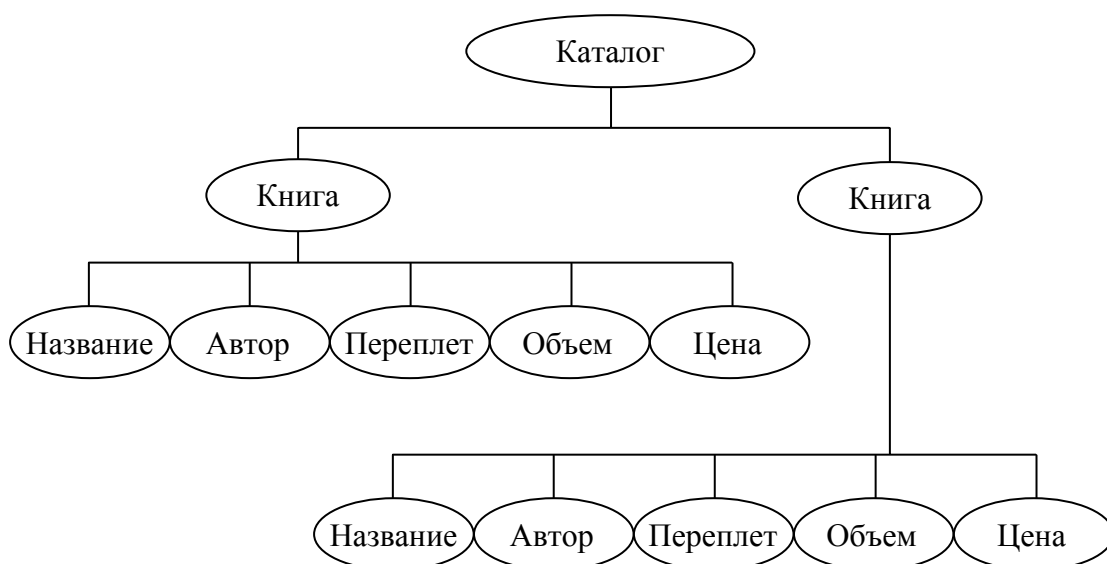


Рис. 1.4. Структура предыдущего XML-документа

Таким образом, с помощью XML вы можете описать иерархическую структуру документа, такого как книга, содержащего части, главы и разделы.

1.4. СОЗДАНИЕ XML-ДОКУМЕНТОВ

Поскольку в XML нет типовых элементов, может показаться, что в нем вообще отсутствуют какие-либо стандарты. Тем не менее язык XML имеет строго определенный синтаксис. Например, в отличие от HTML, каждый элемент XML должен содержать начальный и конечный тег (либо специальный пустой тег, о котором пойдет речь в последующих главах). Любой вложенный элемент должен быть полностью определен внутри элемента, в состав которого он входит.

На деле гибкость в создании ваших собственных элементов требует строгого соблюдения синтаксиса. Это обусловлено тем, что структура XML-документов должна быть понятной для программы, которая обрабатывает и отображает информацию, содержащуюся в этих документах. Строгий синтаксис придает XML-документу предсказуемую форму и облегчает написание программы обработки. Основное назначение языка XML – облегчить работу с документами в Web.

В главе 2 учебно-методического пособия будут затронуты вопросы создания XML-документов в соответствии с синтаксическими

правилами. Вы узнаете, как создавать XML-документ, удовлетворяющий одному из двух уровней синтаксических ограничений. В зависимости от уровня соответствия стандартам документ может быть корректно сформированным (well-formed) либо валидным (valid).

1.5. ОТОБРАЖЕНИЕ XML-ДОКУМЕНТОВ

При отображении HTML-страницы браузер определяет, что элемент H1, например, является заголовком верхнего уровня, и отображает его в соответствующем формате. Это определяется тем, что данный элемент является частью HTML-стандарта. Но каким образом браузер либо другая программа определяет, как обрабатывать и отображать элементы созданного вами XML-документа (такого как BOOK или BINDING в нашем примере), если вы сами составили эти элементы?

Есть три основных способа сообщить браузеру (в частности, Microsoft Internet Explorer 5), как обрабатывать и отображать каждый из созданных вами XML-элементов. Подробнее об этом будет рассказано в главе 3.

1.6. ТАБЛИЦА СТИЛЕЙ

С помощью данного метода вы связываете таблицу стилей с XML-документом. Таблица стилей представляет собой отдельный файл, содержащий инструкции для форматирования индивидуальных XML-элементов. Вы можете использовать либо каскадную таблицу стилей (Cascading Style Sheet – CSS), которая также применяется для HTML-страниц, либо расширяемую таблицу в формате языка стилевых таблиц (Extensible Stylesheet Language – XSL), обладающую значительно более широкими возможностями, нежели CSS, и разработанную специально для XML-документов. Эти методы будут рассмотрены в главах 2, 7 и 10.

1.7. СВЯЗЫВАНИЕ ДАННЫХ

Этот метод требует создания HTML-страницы, связывания с ней XML-документа и установления взаимодействий стандартных

HTML-элементов на странице, таких как SPAN или TABLE, с элементами XML. В дальнейшем HTML-элементы автоматически отображают информацию из связанных с ними XML-элементов.

1.8. НАПИСАНИЕ СЦЕНАРИЯ

В этом методе вы создаете HTML-страницу, связываете ее с XML-документом и имеете доступ к индивидуальным XML-элементам с помощью специально написанного кода сценария (JavaScript или Microsoft Visual Basic Scripting Edition [VBScript]). Браузер воспринимает XML-документ как объектную модель документа (Document Object Model – DOM), состоящую из большого набора объектов, свойств и команд. Написанный код позволяет осуществлять доступ, отображение и манипулирование XML-элементами. Этот метод подробно будет описан в главе 9.

1.9. SGML, HTML И XML

Обобщенный структурированный язык разметки (Structured Generalized Markup Language – SGML) является родоначальником всех языков разметки. Языки HTML и XML образованы из SGML (хотя и различными способами). SGML определяет базовый синтаксис, но дает вам возможность создавать собственные элементы (отсюда термин «обобщенный» в названии языка). Чтобы использовать SGML для описания определенного документа, вы должны продумать соответствующий набор элементов и структуру документа. Например, чтобы описать книгу, вы должны использовать созданные вами элементы с именами BOOK, PART, CHAPTER, INTRODUCTION, A-SECTION, B-SECTION, C-SECTION и т. д.

Набор наиболее употребительных элементов, используемых для описания документа определенного типа, называется SGML-приложением. (SGML-приложение также включает в себя правила, устанавливающие способы организации элементов, а также другие особенности их взаимодействия – о чем пойдет речь в главе 5.) Вы можете определить ваше собственное SGML-приложение, чтобы описать тип документа, с которым вы работаете, либо в теле основной программы должно быть определено SGML-приложение

для описания типовых документов. Наиболее известным примером последнего типа приложений является HTML, который представляет собой SGML-приложение, разработанное в 1991 г. для описания Web-страниц.

Казалось бы, язык SGML вполне подходит для описания Web-документов. Однако разработчики из консорциума W3C посчитали, что он является слишком сложным и фундаментальным, чтобы эффективно представлять информацию в Web. Гибкость и большое обилие средств, поддерживаемых SGML, затрудняет написание программного обеспечения, необходимого для обработки и отображения SGML-информации в Web-браузерах. Следовало бы приспособить часть языка SGML специально для помещения информации в Web. В 1996 г. группа XML Working Group разработала ветвь языка SGML, назвав его расширяемым языком разметки – Extensible Markup Language.

XML является упрощенной версией SGML, приспособленной для Web. Как и SGML, XML дает возможность разрабатывать собственные наборы элементов при описании определенного документа. Как и в SGML, в теле программы может быть определено XML-приложение (или словарь), которое содержит набор наиболее употребительных элементов общего назначения и структуру документа, которая может быть использована для описания документа определенного типа (например, документов, содержащих математические формулы или векторную графику). Об XML-приложениях вы подробнее узнаете далее в этой главе.

Синтаксис XML более простой, чем SGML, что облегчает восприятие XML-документов, а также написание программ браузеров, кодов и Web-страниц для доступа и представления информации документа.

1.10. ЗАМЕНИТ ЛИ XML HTML?

На сегодняшний день ответ на этот вопрос отрицательный. HTML по-прежнему остается основным языком для сообщения браузеру, как отображать информацию в Web.

В Internet Explorer 5 вы можете открывать XML-документы с вложенными таблицами стилей непосредственно браузером, не используя HTML-страницы. Однако в двух других основных методах отображения XML-документов – связывании данных

и DOM-сценариях – отображение XML-документов осуществляется через HTML Web-страницы. (Даже при применении метода таблиц стилей в случае, если вы используете язык XSL, вам потребуется воспользоваться HTML, чтобы сообщить браузеру, каким образом форматировать XML-данные.)

Не заменяя HTML, XML в настоящее время используется в сочетании с ним, существенно расширяя возможности Web-страниц:

- для виртуального представления документов любого типа;
- сортировки, фильтрации, упорядочения, поиска и манипулирования информацией иными способами;
- представления информации в структурированном виде.

Как заявляют сами разработчики, XML был создан для взаимодействия с HTML и совместного с ним использования.

1.11. ОФИЦИАЛЬНЫЕ КОНЦЕПТУАЛЬНЫЕ ЦЕЛИ XML

Ниже представлено десять концепций предназначения и целей применения XML, заявленных в официальной спецификации W3C.

XML должен стать языком прямого использования в Internet. Как вы уже могли понять, XML был разработан главным образом для хранения и распространения информации в Web.

XML будет поддерживать большое количество приложений. Хотя основным его назначением является распространение информации в Web через серверы и программы-браузеры, XML также разработан для использования его другими программами. Например, XML применяется для обмена информацией между финансовыми программами, для распространения и обновления программных продуктов, а также написания голосовых сценариев при доставке информации по телефону.

XML будет совместим с SGML. XML является специализированной ветвью SGML. Преимущество здесь заключается в простоте адаптации программных средств SGML для работы с HTML.

Будет легче писать программы, обрабатывающие XML-документы. Для практического использования XML необходимо, чтобы было достаточно просто писать браузеры и другие программы, обрабатывающие XML-документы. На деле основной причиной выделения XML из SGML была доступность написания программ для обработки XML-документов.

Все перечисленные далее свойства являются в той или иной степени производными этой основной концепции.

Количество дополнительных функций в XML должно быть минимальным, а в идеале – нулевым. Минимальное число дополнительных функций в XML упрощает написание программ для обработки XML-документов. Изобилие дополнительных подключаемых функций в SGML стало основной причиной, обусловившей его практическую непригодность для представления Web-документов. Дополнительные функции SGML требуют переопределения символов-разделителей для тегов (обычно `<and>`) и пропуск конечного тега с целью обнаружения процессором конца элемента. При строгом написании программы обработки SGML-документов необходимо учитывать возможность появления всех дополнительных функций, даже если они редко встречаются.

XML-документы должны быть понятными и ясными для пользователя. XML призван стать универсальным языком (*lingua franca*) для обмена информацией среди пользователей и программ по всему миру. В соответствии с этой концепцией пользователи, а также специализированные программы должны иметь возможность создавать и прочитывать XML-документы. Доступность и прозрачность для пользователя выделяют XML из большинства других форматов, применяемых при построении баз данных и текстовых документов.

Пользователь может легко прочесть XML-документ, поскольку он описан простым текстом и имеет логичную иерархическую структуру в виде дерева. Вы можете упростить XML-документы, назначив информативные имена для элементов, атрибутов и объектов, а также добавив полезные комментарии.

Разработка XML должна быть завершена достаточно быстро. XML станет общепринятым стандартом лишь в том случае, если программисты и пользователи примут его. Следует создать этот стандарт до того, как общество примет альтернативные стандарты, которые все быстрее создаются компаниями-разработчиками программного обеспечения.

Язык XML должен быть формальным и кратким. Спецификация XML написана на формальном языке, используемом для представления компьютерных языков. Этот формальный язык, хотя и достаточно сложен для восприятия, лишен двусмысленности и существенно облегчает написание XML-документов, а в особенности программ для их обработки.

XML-документы будет проще создавать. При практическом использовании XML как языка разметки для Web-документов упрощается не только написание обрабатывающих программ, но и процесс создания самих XML-документов.

Сжатой форме при XML-разметке придается минимальное значение. В соответствии с положением о том, что XML-документ должен быть ясным и понятным для пользователя, XML-разметка не должна быть излишне сжатой, чтобы не вступать в противоречие с указанной целью.

1.12. СТАНДАРТНЫЕ XML-ПРИЛОЖЕНИЯ

Вы можете использовать XML не только для описания отдельного документа. Индивидуальный пользователь, компания или комитет по стандартам может определить необходимый набор элементов XML и структуру документа, которые будут применяться для особого класса документов. Подобный набор элементов и описание структуры документа называют XML-приложением или XML-словарем.

Например, организация может определить XML-приложение для создания документов, описывающих молекулярные структуры, людские ресурсы, мультимедиа-презентации или содержащих векторную графику. В конце подглавы будет приведен список некоторых уже созданных общеупотребительных XML-приложений, либо приложений, создание которых планируется.

XML-приложение обычно определяется созданием описателя типа документа (DTD), который является допустимым компонентом XML-документа. DTD построен по схеме базы данных: он устанавливает и определяет имена элементов, которые могут быть использованы в документе, порядок, в котором элементы могут появляться, доступные к применению атрибуты элементов и другие особенности документа. Для практического использования XML-приложения вы обычно включаете его DTD в ваш XML-документ; наличие DTD в документе ограничивает круг элементов и структур, которые вы будете использовать, вследствие чего ваш документ отвечает стандартам данного приложения. Описания XML-документов, рассмотренных ранее в этой главе, не включали DTD. О том, как задавать и использовать DTD, вы узнаете в главе 5.

Преимущества применения стандартных XML-приложений при разработке ваших документов состоят в том, что вы можете совместно использовать документы со всеми другими пользователями приложения, а документ может обрабатываться и отображаться с помощью программного обеспечения, которое уже создано для данного приложения.

Кроме XML-приложений, для описания определенных классов документов имеется несколько XML-приложений, которые вы можете применять внутри XML-документа любого типа. Эти приложения облегчают создание документа и улучшают его качество. Ниже приведены примеры таких приложений.

Extensible Stylesheet Language (XSL) дает вам возможность создавать мощные стилевые таблицы с использованием синтаксиса XML.

XML Schema позволяет разрабатывать подробные схемы для ваших XML-документов с использованием стандартного синтаксиса XML, что является более мощной альтернативой применения DTD.

XML Linking Language (XLink) дает возможность связывать ваши XML-документы. Он поддерживает множественные целевые ссылки и другие полезные функции, обеспечивая большую свободу по сравнению с механизмом организации ссылок в HTML.

XML Pointer Language (XPointer) позволяет определять гибкие целевые ссылки. При совместном использовании XPointer и XLink вы можете организовывать ссылки на любое место в целевом документе, а не только переходы к специально выделенным пунктам.

Об XSL будет рассказано в главе 10. Другие XML-приложения еще не доведены до готовности и не рассматриваются в этом курсе. (XLink и XPointer не поддерживаются в Internet Explorer 5.)

Как видите, XML является не только полезным инструментом для описания документов, но и служит основой для построения приложений и расширений, которые могут оказаться востребованными по мере развития Internet.

1.13. РЕАЛЬНОЕ ИСПОЛЬЗОВАНИЕ XML

Хотя концепция XML весьма интересна, у вас может возникнуть вопрос, как его использовать на практике. В этой подглаве приведен перечень примеров такого применения XML, как уже

широко используемых, так и перспективных. Если имеются соответствующие XML-приложения для практического использования, они будут приведены в скобках. Например, вы сможете узнать, что XML-приложение MathML позволит вам форматировать математические формулы:

1) работа с базами данных. Подобно традиционным базам данных XML может быть использован для присвоения метки каждому полю информации внутри каждой записи базы данных. (Например, можно пометить каждое имя, адрес и номер телефона внутри записей списка адресов.) После этого вы сможете отображать данные различными способами и организовывать поиск, сортировку, фильтрацию и иную обработку данных;

2) структурирование документов. Иерархическая структура XML-документов идеально подходит для разметки структуры таких документов, как романы, научные труды, пьесы. Например, вы можете использовать XML для разметки пьесы на акты, сцены, размечать действующих лиц, сюжетные линии, декорации и т. д. XML-разметка дает возможность программам отображать или распечатывать документ в необходимом формате; находить, извлекать или манипулировать информацией в документе; генерировать оглавления, резюме и аннотации; обрабатывать информацию иными способами;

3) работа с векторной графикой (Vector Markup Language – VML);

4) мультимедиа-презентации (Synchronized Multimedia Integration Language – SMIL, HTML Timed Interactive Multimedia Extensions – HTML + TIME);

5) описание каналов. Каналы представляют собой Web-страницы, которые автоматически рассылаются подписчикам (Channel Definition Format – CDF);

6) описание программных пакетов и их взаимосвязей. Такие описания обеспечивают распространение и обновление программных продуктов в сети (Open Software Description – OSD);

7) взаимодействие приложений через Web с использованием XML-сообщений. Эти сообщения являются независимыми от операционных систем, объектных моделей и компьютерных языков (Simple Object Access Protocol – SOAP);

8) отправка электронных бизнес-карт через e-mail;

9) обмен финансовой информацией. Обмен информацией в открытом и понятном формате осуществляется между финансовыми программами (такими как Quicken и Microsoft Money) и финансовыми

институтами (банками, общественными фондами) (Open Financial Exchange – OFX);

10) создание, управление и использование сложных цифровых форм для коммерческих Internet-транзакций. Подобные формы могут включать оцифрованные подписи, которые делают их признанными юридически (Extensible Forms Description Language – XFDL);

11) обмен запросами по приему на работу и резюме (Human Resource Management Markup Language – HRMML);

12) форматирование математических формул и научной информации в Web (Mathematical Markup Language – MathML);

13) описание молекулярных структур (Chemical Markup Language – CML);

14) кодирование и отображение информации о ДНК, РНК и цепочках (Bioinformatic Sequence Markup Language – BSML);

15) кодирование генеалогических данных (Genealogical Data Markup Language – GeDML);

16) обмен астрономическими данными (Astronomical Markup Language – AML);

17) создание музыкальных партитур (Music Markup Language – MusicML);

18) работа с голосовыми сценариями для доставки информации по телефону. Голосовые сценарии могут быть использованы, например, для генерирования голосовых сообщений, справок о наличии товаров и прогнозов погоды (VoxML);

19) обработка и доставка информации курьерскими службами. Служба Federal Express, например, уже использует XML для этих целей;

20) представление рекламы в прессе в цифровом формате (Ad Markup);

21) заполнение юридических документов и электронный обмен юридической информацией (XML Court Interface – XCI);

22) кодирование прогнозов погоды (Weather Observation Markup Format – OMF);

23) обмен страховой информацией;

24) обмен новостями и информацией с использованием открытых Web-стандартов (XMLNews);

25) представление религиозной информации и разметка текстов богослужений (Theological Markup Language – ThML, Liturgical Markup Language – LitML).

? ЗАДАНИЯ

1. Ответьте на следующие вопросы:

- Что представляет собой язык XML?
- Какие проблемы решает язык XML?
- Для чего XML используется в сочетании с HTML?

2. Расскажите о следующих методах:

- таблица стилей;
- связывание данных;
- написание сценария.

3. Назовите концепции предназначения и целей применения XML.

4. Приведите примеры использования XML.

Глава 2. РАБОТА С XML-ДОКУМЕНТАМИ

2.1. СОЗДАНИЕ И ОТОБРАЖЕНИЕ ВАШЕГО ПЕРВОГО XML-ДОКУМЕНТА

В этой главе вы получите представление о процессе создания и отображения XML-документа в Web-браузере. Сначала вы создадите простой XML-документ, исследуете его структуру и познакомитесь с основными правилами создания корректно сформированного XML-документа. Затем вы узнаете, как отобразить этот документ в браузере Microsoft Internet Explorer 5 путем создания и присоединения простой стилевой таблицы, которая сообщает браузеру, как форматировать элементы в документе.

2.1.1. Создание XML-документа

Поскольку описание XML-документа представляет собой простой текст, вы можете создать его, используя ваш любимый текстовый редактор, например редактор Notepad, входящий в состав Microsoft Windows. Еще лучше воспользоваться редактором, в котором предусмотрена возможность анализа исходных кодов, например текстовым редактором Microsoft Visual Studio, рассчитанным на работу с Microsoft Visual C++ и другими приложениями Visual Studio.

XML-документ состоит из двух основных частей: пролога и элемента Документ (его также называют корневым элементом).

2.1.2. Пролог

В данном примере документа пролог состоит из трех строк:

```
<?xml version="1.0"?>
```

```
<!-- File Name: Inventory.xml -->
```

Первая строка представляет собой объявление XML, указывающее на то, что это XML-документ, и содержащее номер версии. Объявление XML не является обязательным, хотя спецификация требует его включения. Если вы включаете XML-объявление, оно должно находиться в начале документа.

Вторая строка пролога состоит из пробела. С целью улучшения внешнего вида документа вы можете вставлять любое количество пустых строк между элементами пролога. При обработке они будут игнорироваться.

Третья строка пролога представляет собой комментарий. Добавление комментариев в XML-документ не обязательно, но позволяет сделать его более понятным. Комментарий начинается с символов `<!--` и заканчивается символами `-->`. Между этими двумя группами символов вы можете поместить любой текст (за исключением `-->`); XML-процессор проигнорирует его.

⇒ *Примечание.* Все составляющие пролога, упомянутые в этом пункте, будут подробно описаны далее в последующих главах.

Пролог может также содержать следующие необязательные компоненты:

- объявление типа документа, определяющее тип и структуру документа. Объявление типа документа должно следовать после XML-объявления;
- одна или несколько инструкций по обработке, содержащих информацию о порядке проходов при обработке приложения XML-процессором. Далее в этой главе вы познакомитесь с инструкцией по обработке для связывания таблицы стилей с XML-документом.

⇒ *Примечание.* XML-процессор – это программный модуль, считывающий XML-документ и обеспечивающий доступ к его содержимому. Он также предоставляет этот доступ другим программным модулям или приложениям, которые манипулируют и отображают содержимое документа. Если вы отображаете XML-документ в Internet Explorer 5, браузер включает в себя как XML-процессор, так и приложение. (Если для отображения XML-документа вы используете HTML и сценарий (скрипт-код), то при этом самостоятельно создаете часть приложения.) Обратите внимание, что термин «приложение» в данном случае отличен от термина, применяемого для обозначения XML-приложения (или словаря) как целевого набора элементов и структуры документа, которые используются для описания документов определенного типа.

2.1.3. Элемент Документ

Второй основной частью XML-документа является единый элемент Документ, или корневой элемент, который в свою очередь содержит дополнительные элементы.

В XML-документе элементы определяют его логическую структуру и несут в себе информацию, содержащуюся в документе (в нашем примере это информация о книгах, такая как название, автор, цена). Типовой элемент состоит из начального тега, содержимого элемента и конечного тега. Содержимым элемента могут быть символьные данные, другие (вложенные) элементы, либо сочетание данных и вложенных элементов.

В рассматриваемом примере (см. листинг 1.2 на с. 15) элемент Документ – INVENTORY. Его начальный тег – <INVENTORY>, конечный тег – </INVENTORY>, а содержимое – пять вложенных элементов BOOK.

⇒ *Примечание.* Текст в XML-документе представляет собой перемежающиеся символьные данные и данные, относящиеся к разметке. Разметка – это текст, ограниченный разделителями и описывающий структуру документа, а именно: начальный и конечный теги элемента, теги пустого элемента, объявления типа документа, инструкции по обработке, ограничители раздела CDATA, символьные ссылки, ссылки на примитивы (entity). Остальной текст представляет собой символьные данные – реальное информационное содержимое документа (в нашем примере это названия, фамилии авторов, цена и другая информация о книге).

⇒ *Примечание.* Элемент Документ в XML-документе похож на элемент BODY на HTML-странице, за исключением того, что вы можете присвоить ему любое допустимое имя.

В свою очередь, каждый элемент BOOK содержит ряд вложенных элементов (рис. 2.1).

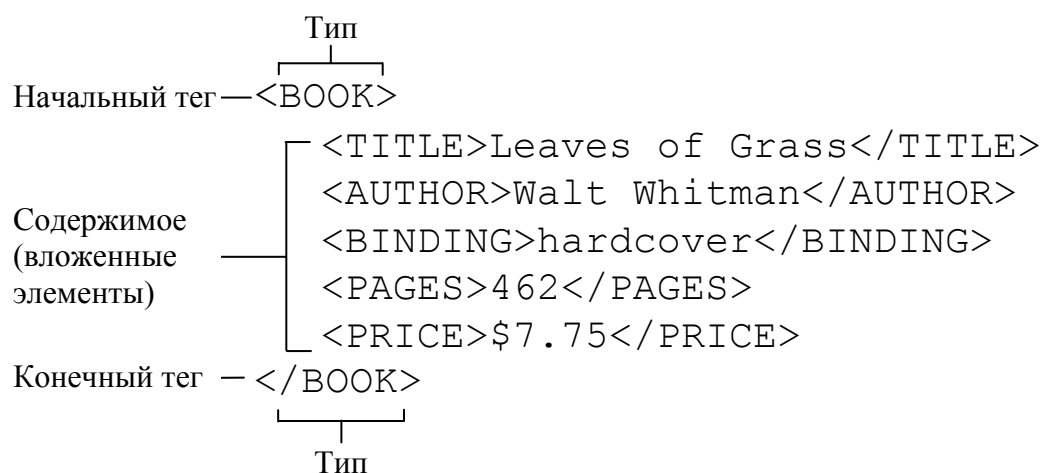


Рис. 2.1. Вложенные элементы BOOK

⇒ *Примечание.* Имя, которое содержится в начальном и конечном теге, есть тип элемента.

Каждый из элементов, вложенных в элемент BOOK, например элемент TITLE, содержит только символьные данные (рис. 2.2).



Рис. 2.2. Символьные данные элемента TITLE

Далее вы узнаете о дополнительных элементах XML-документа и о включении атрибутов в начальный тег элемента.

2.2. НЕКОТОРЫЕ БАЗОВЫЕ ПРАВИЛА XML

Ниже приведено несколько основных правил создания форматированного XML-документа. Форматированный документ соответствует минимальному набору правил, обеспечивающих возможность обработки документа браузером или другой программой. Документ, составленный вами ранее в этой главе, является примером форматированного XML-документа, удовлетворяющего этим правилам.

Документ должен иметь только один элемент верхнего уровня (элемент Документ, или корневой элемент). Все другие элементы должны быть вложены в элемент верхнего уровня.

Элементы должны быть вложены упорядоченным образом, т. е. если элемент начинается внутри другого элемента, он должен и заканчиваться внутри этого элемента.

Каждый элемент должен иметь начальный и конечный теги. В отличие от HTML, в XML не разрешается опускать конечный тег – даже в том случае, когда браузер в состоянии определить, где заканчивается элемент. (В главе 3 вы познакомитесь с усеченной нотацией, которую можно применять для пустых элементов, т. е. элементов, не имеющих содержимого.)

Имя типа элемента в начальном теге должно в точности соответствовать имени в соответствующем конечном теге.

Имена типов элементов чувствительны к регистру, в котором они набраны. В действительности весь текст внутри XML-разметки является чувствительным к регистру. Например, следующее описание элемента является неправильным, поскольку имя типа элемента в начальном теге не соответствует имени типа в конечном теге:

```
<TITLE>Leaves of grass</Title>
```

2.3. ОТОБРАЖЕНИЕ XML-ДОКУМЕНТА

Вы можете открыть XML-документ непосредственно через Internet Explorer 5, точно так же, как вы бы открыли HTML Web-страницу.

Если XML-документ не содержит связи с таблицей стилей, Internet Explorer 5 помечает различные составные части документа различным цветом, чтобы облегчить их распознавание, а также представляет элемент Документ в виде иерархического дерева с возможностью свертывания и развертывания структуры и просмотра с меньшей или большей степенью детализации.

Если же XML-документ имеет связь с таблицей стилей, Internet Explorer 5 отобразит только символьные данные из элементов документа, отформатировав их в соответствии с правилами, установленными в таблице стилей. Вы можете использовать либо таблицу каскадных стилей (CSS-таблицу, аналогичную той, которая используется для HTML-страниц), либо XSL-таблицу стилей, которая является более мощным инструментом и строится в соответствии с синтаксисом, принятым для XML. Такие таблицы могут использоваться исключительно для XML-документов.

2.4. ОБНАРУЖЕНИЕ ОШИБОК XML В INTERNET EXPLORER 5

Прежде чем Internet Explorer 5 отобразит ваш XML-документ, его встроенный синтаксический XML-анализатор (parser) просматривает содержимое документа. Если он обнаружит ошибку, Internet Explorer 5 отобразит страницу с сообщением об ошибке, не предпринимая попытки отобразить документ. Internet Explorer 5 отобразит

страницу с сообщением об ошибке независимо от того, связан ли XML-документ с таблицей стилей.

⇒ *Примечание.* Синтаксический XML-анализатор является составной частью XML-процессора, который сканирует XML-документ, анализирует его структуру и обнаруживает синтаксические ошибки.

⇒ *Примечание.* Когда вы открываете XML-документ непосредственно в Internet Explorer 5, транслятор проверяет лишь соответствие документа формальным правилам построения (корректность формы) и в случае обнаружения несоответствия выдает сообщение об ошибке. Полное соответствие правилам для документа (валидность) не проверяется. Даже в случае, если вы не связываете таблицу стилей с XML-документом, Internet Explorer 5 использует для отображения документа имеющуюся по умолчанию таблицу стилей; именно поэтому в описании ошибки упоминается использование XSL-таблицы стилей («using XSL style sheet»). О XSL-таблицах стилей вы узнаете в главе 10.

ЗАДАНИЯ

1. Создайте XML-документ

Откройте новый файл в вашем текстовом редакторе и введите текст XML-документа (см. листинг 2.2 на с. 36–37).

При желании можно опустить некоторые элементы типа BOOK. Вам не обязательно набирать все восемь – достаточно будет трех или четырех.

Воспользуйтесь командой Save (Сохранить) текстового редактора для сохранения документа на вашем жестком диске, присвоив ему имя Inventory.xml.

2. Отобразите XML-документ без таблицы стилей

В Windows Explorer (Проводник) или в окне папки дважды щелкните на имени файла Inventory.xml, который вы сохранили в предыдущем упражнении.

Попробуйте изменить степень детализации представления элементов документа. Щелкните на символе знака минус (–) слева от начального тега, чтобы свернуть элемент, либо на знаке плюс (+) рядом со свернутым элементом, чтобы развернуть его. Например, щелкнув на знаке минус (–) рядом с элементом INVENTORY, вы получите то же, что представлено на рис. 2.3.

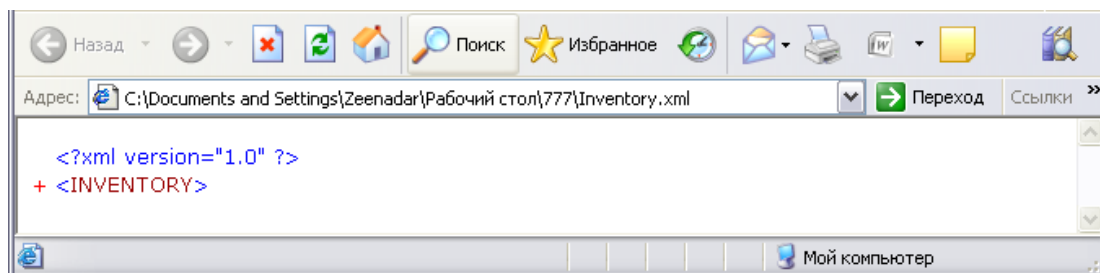


Рис. 2.3. Отображение в Internet Explorer 5 представления элементов документа

3. Создайте искусственную ошибку

В вашем текстовом редакторе откройте документ Inventory.xml, созданный вами в предыдущем упражнении. Измените первый элемент

TITLE с <TITLE>The Adventures of Huckleberry Finn</TITLE>

на

<TITLE>The Adventures of Huckleberry Finn</Title>

4. Сохраните внесенные изменения

В Windows Explorer (Проводник) или в окне папки дважды щелкните на имени файла Inventory.xml. Вместо того, чтобы отобразить XML-документ, Internet Explorer 5 теперь отобразит следующую страницу с сообщением об ошибке (рис. 2.4).

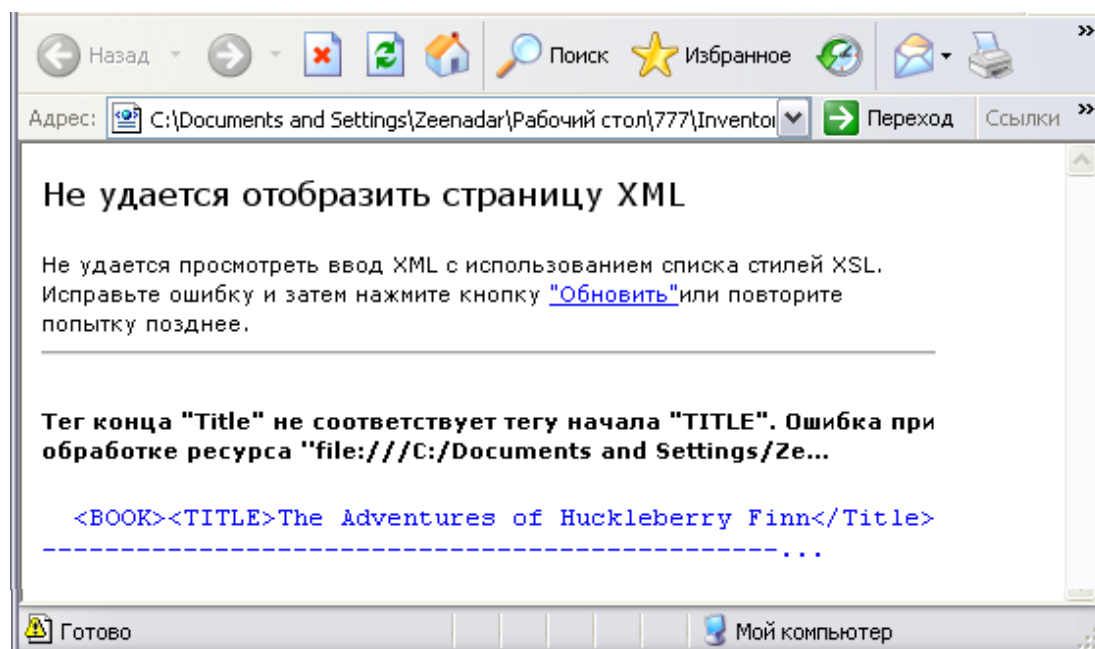


Рис. 2.4. Отображение в Internet Explorer 5 сообщения об ошибке

Поскольку вы еще будете работать с Inventory.xml в этой главе, вам теперь нужно восстановить конечный тег для первого элемента TITLE, вернув ему первоначальный вид (/TITLE), а затем повторно сохранить документ.

5. Отобразите XML-документ с использованием таблицы каскадных стилей

Откройте новый, пустой текстовый файл в вашем текстовом редакторе и заполните CSS-таблицу.

С помощью команды Save (Сохранить) вашего текстового редактора сохраните таблицу стилей на жестком диске, задав имя файла Inventory01.css. Созданная вами CSS-таблица сообщает Internet Explorer 5, каким образом форматировать символьные данные элементов:

- отображать каждый элемент BOOK с отступом сверху в 12 пт (margin-top:12pt) и с переводом строки сверху и снизу (display:block), используя размер шрифта 10 пт (font-size:10pt);
- отображать каждый элемент TITLE курсивом (font-style:italic);
- отображать каждый элемент AUTHOR полужирным (font-weight:bold).

Листинг 2.1. Форматирование символьных данных

```
/* File Name: Inventory01.css */
BOOK
    {display:block;
      margin-top:12pt;
      font-size:10pt}
TITLE
    {font-style:italic}
AUTHOR
    {font-weight:bold}
```

В вашем текстовом редакторе откройте документ Inventory.xml, созданный вами в предыдущем упражнении. Добавьте в конце пролога документа (непосредственно над элементом INVENTORY) следующую инструкцию по обработке:

```
<?xml-stylesheet type="text/css" href="Inventory01.css"?>
```

Эта инструкция по обработке устанавливает связь между созданной вами CSS-таблицей и XML-документом. В результате при открытии документа в Internet Explorer 5 браузер отобразит содержимое документа в соответствии с инструкциями, записанными в таблице стилей.

Чтобы отразить новое имя файла, которое вы собираетесь присвоить, измените комментарий в начале документа

```
<!-- File Name: Inventory.xml -->
```

на

```
<!-- File Name: Inventory01.xml -->
```

Воспользуйтесь командой Save As (Сохранить как) вашего текстового редактора, чтобы сохранить копию измененного документа под именем Inventory01.xml. Проверьте, чтобы этот файл был сохранен в той же папке, что и файл Inventory01.css.

Листинг 2.2. Сохранение копии документа

```
<?xml version="1.0"?>
<!-- File Name: Inventory01.xml -->
<?xml-stylesheet type="text/css" href="Inventory01.css"?>
<INVENTORY>
  <BOOK>
    <TITLE>The Adventures of Huckleberry Finn</TITLE>
    <AUTHOR>Mark Twain</AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>298</PAGES>
    <PRICE>$5.49</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>Leaves of Grass</TITLE>
    <AUTHOR>Walt Whitman</AUTHOR>
    <BINDING>hardcover</BINDING>
    <PAGES>462</PAGES>
    <PRICE>$7.75</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>The Legend of Sleepy Hollow</TITLE>
    <AUTHOR>Washington Irving</AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>98</PAGES>
    <PRICE>$2.95</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>The Marble Faun</TITLE>
    <AUTHOR>Nathaniel Hawthorne</AUTHOR>
    <BINDING>trade paperback</BINDING>
    <PAGES>473</PAGES>
    <PRICE>$10.95</PRICE>
  </BOOK>
```

```

<BOOK>
  <TITLE>Moby-Dick</TITLE>
  <AUTHOR>Herman Melville</AUTHOR>
  <BINDING>hardcover</BINDING>
  <PAGES>724</PAGES>
  <PRICE>$9.95</PRICE>
</BOOK>
<BOOK>
  <TITLE>The Portrait of a Lady</TITLE>
  <AUTHOR>Henry James</AUTHOR>
  <BINDING>mass market paperback</BINDING>
  <PAGES>256</PAGES>
  <PRICE>$4.95</PRICE>
</BOOK>
<BOOK>
  <TITLE>The Scarlet Letter</TITLE>
  <AUTHOR>Nathaniel Hawthorne</AUTHOR>
  <BINDING>trade paperback</BINDING>
  <PAGES>253</PAGES>
  <PRICE>$4.25</PRICE>
</BOOK>
<BOOK>
  <TITLE>The Turn of the Screw</TITLE>
  <AUTHOR>Henry James</AUTHOR>
  <BINDING>trade paperback</BINDING>
  <PAGES>384</PAGES>
  <PRICE>$3.35</PRICE>
</BOOK>
<BOOK>
  <TITLE>The Portrait of a Lady</TITLE>
  <AUTHOR>Henry James</AUTHOR>
  <BINDING>mass market paperback</BINDING>
  <PAGES>256</PAGES>
  <PRICE>$4.95</PRICE>
</BOOK>
<BOOK>
  <TITLE>The Scarlet Letter</TITLE>
  <AUTHOR>Nathaniel Hawthorne</AUTHOR>
  <BINDING>trade paperback</BINDING>
  <PAGES>253</PAGES>
  <PRICE>$4.25</PRICE>
</BOOK>
<BOOK>
  <TITLE>The Turn of the Screw</TITLE>
  <AUTHOR>Henry James</AUTHOR>
  <BINDING>trade paperback</BINDING>
  <PAGES>384</PAGES>
  <PRICE>$3.35</PRICE>
</BOOK>
</INVENTORY>

```

В Windows Explorer (Проводник) или в окне папки дважды щелкните на файле Inventory01.xml, чтобы открыть документ.

Internet Explorer 5 откроет документ Inventory01.xml и отобразит его в соответствии с правилами связанной таблицы стилей, как показано на рис. 2.5.

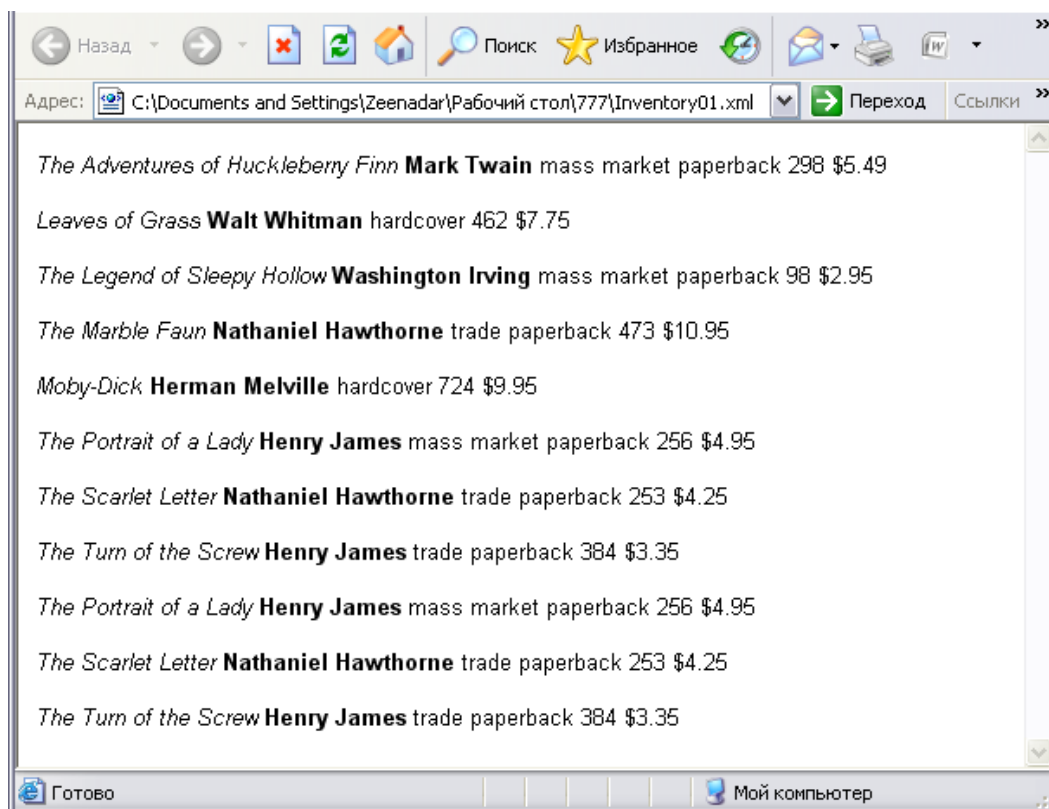


Рис. 2.5. Отображение в Internet Explorer 5 файла Inventory01.xml

Откройте новый, пустой текстовый файл в вашем текстовом редакторе и введите измененную CSS-таблицу.

Воспользуйтесь командой Save (Сохранить) вашего текстового редактора, чтобы сохранить новую таблицу стилей на жестком диске, присвоив ей имя файла Inventory02.css.

Созданная вами модифицированная таблица стилей сообщает Internet Explorer 5, каким образом форматировать символьные данные элементов:

- отображать каждый элемент BOOK с отступом сверху в 12 пт (margin-top:12pt) и с переводом строки сверху и снизу (display:block), используя размер шрифта 10 пт (font-size:10pt);

- отображать каждый из элементов TITLE, AUTHOR, BINDING и PRICE в отдельной строке (display:block);

- отображать элемент TITLE шрифтом размером 12 пт (font.size:12pt), полужирным (font-weight:bold), курсивом (font-style:italic);
- отступ слева для каждого из элементов AUTHOR, BINDING и PRICE на 15 пт (margin-left:15pt);
- отображать элемент AUTHOR полужирным (font-weight:bold);
- не отображать элемент PAGES (display:none).

Листинг 2.3. Форматирование символьных данных с раскрытием вложенных элементов

```
/* File Name: Inventory02.css */
BOOK
    {display:block;
      margin-top:12pt;
      font-size:10pt}
TITLE
    {display:block;
      font-size:12pt;
      font-weight:bold;
      font-style:italic}
AUTHOR
    {display:block;
      margin-left:15pt;
      font-weight:bold}
BINDING
    {display:block;
      margin-left:15pt}
PAGES
    {display:none}
PRICE
    {display:block;
      margin-left:15pt}
```

В вашем текстовом редакторе откройте документ Inventory.xml. Добавьте в конце пролога документа (над элементом INVENTORY) следующую инструкцию по обработке:

```
<?xml-stylesheet type="text/css" href="Inventory02.css"?>
```

Эта инструкция по обработке устанавливает связь между новой CSS-таблицей, созданной вами, и XML-документом.

Чтобы отразить новое присвоенное вами имя файла, измените комментарий в начале документа

```
<!-- File Name: Inventory.xml -->
```

на

```
<!-- File Name: Inventory02.xml -->
```

Воспользуйтесь командой Save As (Сохранить как), чтобы сохранить копию измененного документа под именем Inventory02.xml. Проверьте, чтобы этот файл был сохранен в той же папке, что и файл Inventory02.css.

В Windows Explorer (Проводник) или в окне папки дважды щелкните на имени файла Inventory02.xml, чтобы открыть его.

Internet Explorer 5 откроет документ Inventory02.xml и отобразит его в соответствии с правилами, установленными в связанной таблице стилей Inventory02.css (рис. 2.6).

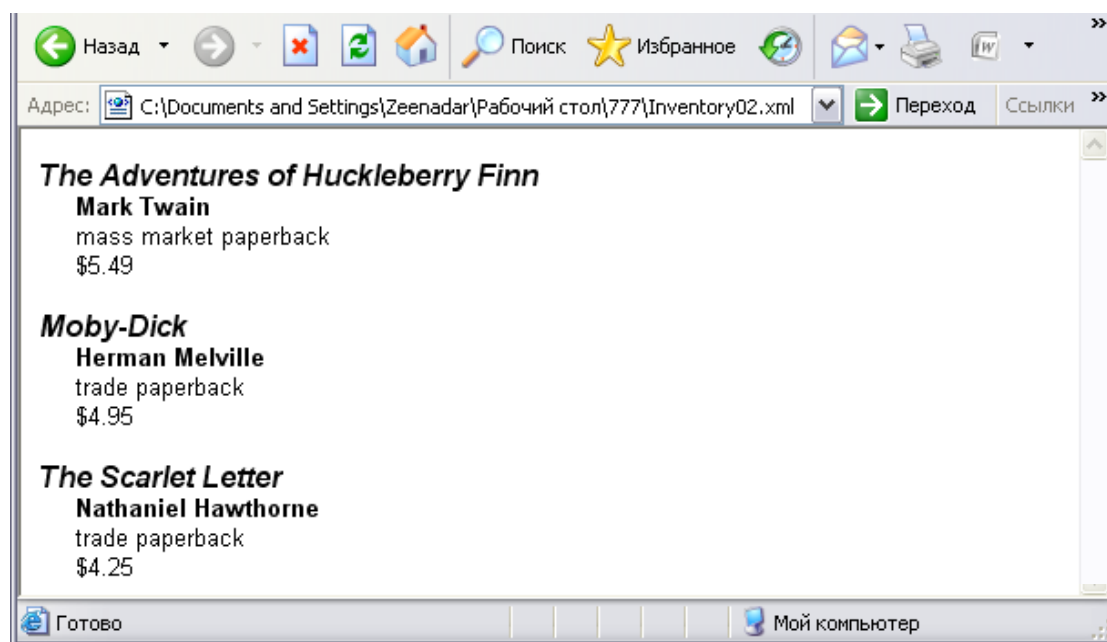


Рис. 2.6. Отображение в Internet Explorer 5 файла Inventory02.xml

Здесь представлено лишь три книги; осуществив прокрутку вниз, вы увидите остальные.

Глава 3. ЭЛЕМЕНТЫ И АТТРИБУТЫ XML-ДОКУМЕНТОВ

3.1. СОЗДАНИЕ КОРРЕКТНО СФОРМИРОВАННЫХ XML-ДОКУМЕНТОВ

В этой главе вы познакомитесь с основными приемами создания корректно сформированных (well-formed) XML-документов. Корректно сформированным называется документ, отвечающий минимальному набору критериев соответствия для XML-документа. Когда вы создаете корректно сформированный XML-документ, вы можете добавлять элементы и вводить данные непосредственно в ваш документ, как вы это делаете при создании HTML-документов.

3.1.1. Составные части корректно сформированного XML-документа

В главе 2 вы узнали, что XML-документ состоит из двух основных частей: пролога и элемента Документ (корневого элемента). Помимо этого, вслед за элементом Документ корректно сформированный XML-документ может содержать комментарии, инструкции по обработке, а также пробелы.

Номер версии в XML-объявлении в начале пролога документа может быть заключен как в одинарные, так и в двойные кавычки. Строки в кавычках в XML-разметке носят название «литерал». Таким образом, обе приведенные ниже записи являются допустимыми:

```
<?xml version='1.0'?>  
<?xml version="1.0"?>
```

XML-объявление в примере также включает в себя объявление документа автономным (standalone = «yes»). Это объявление может использоваться в некоторых XML-документах с целью упростить обработку документа.

В рассматриваемом примере имеется комментарий в прологе, а также другой комментарий, следующий за элементом Документ.

Документ содержит две пустые строки в прологе и две пустые строки в разделе, следующем за элементом Документ. Пустая строка состоит из одного или нескольких пробелов, символа табуляции,

возврата каретки или перевода строки. Чтобы улучшить внешний вид и восприятие XML-документа, вы можете свободно добавлять пробелы и пустые строки между элементами XML-разметки, такими как начальные и конечные теги, комментарии и инструкции по обработке, а также во многих случаях внутри элементов разметки – например, пробел между «yes» и в конце XML-объявления в рассматриваемом примере. Процессор просто игнорирует пропуски, если только они не находятся внутри элемента, непосредственно содержащего символьные данные.

Документ включает в себя *sine qua non* (обязательную часть) XML-документа – элемент Документ. Создание элемента Документ и вложенных в него элементов является основным предметом рассмотрения в этой главе.

⇒ *Примечание.* Валидный документ должен содержать один дополнительный компонент, не включенный в листинг рассматриваемого в примере документа: объявление типа документа, которое может быть помещено в любое место внутри пролога, но вне других элементов разметки и после XML-объявления. Объявление типа документа задает структуру валидного XML-документа.

3.1.2. Наименьший XML-документ

Пролог рассматриваемого XML-документа содержит примеры каждого из разрешенных внутри пролога включений. Следовательно, и сам пролог является необязательным, что подтверждается следующим минимальным документом, который содержит только элемент Документ, в соответствии с XML-стандартом для корректно сформированного документа:

```
<minimal>A minimalist document.</minimal>
```

В Internet Explorer 5 этот документ будет отображен, как показано на рис. 3.1.

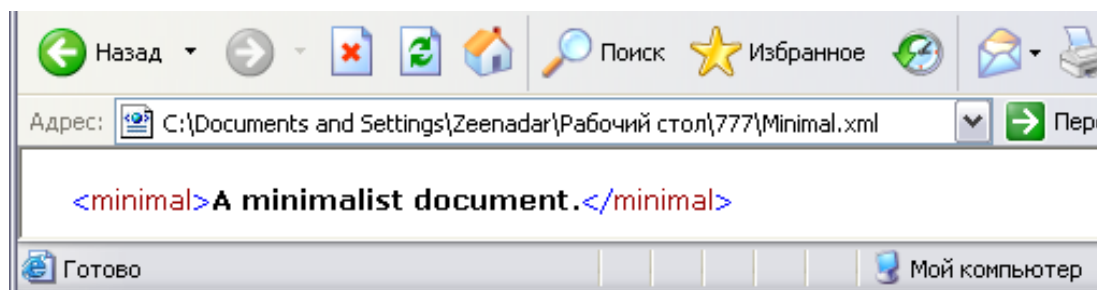


Рис. 3.1. Отображение в Internet Explorer 5 корректно сформированного документа

Заметим, однако, что все эти включения не являются обязательными (хотя в спецификации XML заявлено, что вам следует включать XML-объявление).

3.1.3. Добавление элементов в документ

Элементы в XML-документе содержат фактическую информацию, имеющуюся в документе (для листинга 3.1, например, это названия книг, фамилии авторов, цена и т. д.), а также отражают логическую структуру информации.

Листинг 3.1. Parts.xml

```
<?xml version='1.0' standalone='yes'?>
<!-- File Name: Parts.xml -->
<?xml-stylesheet type="text/css" href="Inventory01.css"?>
<INVENTORY>
  <BOOK>
    <TITLE>The Adventures of Huckleberry Finn</TITLE>
    <AUTHOR>Mark Twain</AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>298</PAGES>
    <PRICE>$5.49</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>Leaves of Grass</TITLE>
    <AUTHOR>Walt Whitman</AUTHOR>
    <BINDING>hardcover</BINDING>
    <PAGES>462</PAGES>
    <PRICE>$7.75</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>The Legend of Sleepy Hollow</TITLE>
    <AUTHOR>Washington Irving</AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>98</PAGES>
    <PRICE>$2.95</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>The Marble Faun</TITLE>
    <AUTHOR>Nathaniel Hawthorne</AUTHOR>
    <BINDING>trade paperback</BINDING>
    <PAGES>473</PAGES>
    <PRICE>$10.95</PRICE>
  </BOOK>
```

```

<BOOK>
  <TITLE>Moby-Dick</TITLE>
  <AUTHOR>Herman Melville</AUTHOR>
  <BINDING>hardcover</BINDING>
  <PAGES>724</PAGES>
  <PRICE>$9.95</PRICE>
</BOOK>
<BOOK>
  <TITLE>The Portrait of a Lady</TITLE>
  <AUTHOR>Henry James</AUTHOR>
  <BINDING>mass market paperback</BINDING>
  <PAGES>256</PAGES>
  <PRICE>$4.95</PRICE>
</BOOK>
<BOOK>
  <TITLE>The Scarlet Letter</TITLE>
  <AUTHOR>Nathaniel Hawthorne</AUTHOR>
  <BINDING>trade paperback</BINDING>
  <PAGES>253</PAGES>
  <PRICE>$4.25</PRICE>
</BOOK>
<BOOK>
  <TITLE>The Turn of the Screw</TITLE>
  <AUTHOR>Henry James</AUTHOR>
  <BINDING>trade paperback</BINDING>
  <PAGES>384</PAGES>
  <PRICE>$3.35</PRICE>
</BOOK>
</INVENTORY>
<!--      Comments, processing instructions, and white space
          can also appear after the document element. -->
<?MyApp Parm1="value 1" Parm2="value 2"?>

```

Элементы организованы в иерархическую древовидную структуру, в которой одни элементы вложены в другие. Документ должен иметь один и только один элемент верхнего уровня – элемент Документ, или корневой элемент, а все другие элементы должны быть вложены в него. Представленный ниже XML-документ является корректно сформированным.

Листинг 3.2. Корректно сформированный документ

```

<?xml version="1.0"?>
<!-- Корректно сформированный XML-документ. -->
<INVENTORY>
  <BOOK>

```

```

        <TITLE>The Adventures of Huckleberry Finn</TITLE>
        <AUTHOR>Mark Twain</AUTHOR>
        <BINDING>mass market paperback</BINDING>
        <PAGES>298</PAGES>
        <PRICE>$5.49</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>Leaves of Grass</TITLE>
        <AUTHOR>Walt Whitman</AUTHOR>
        <BINDING>hardcover</BINDING>
        <PAGES>462</PAGES>
        <PRICE>$7.75</PRICE>
    </BOOK>
</INVENTORY>

```

А вот следующий документ не является корректно сформированным.

Листинг 3.3. Некорректно сформированный документ

```

<?xml version="1.0"?>
<!-- Это некорректно сформированный документ. -->
<BOOK>
    <TITLE>The Adventures of Huckleberry Finn</TITLE>
    <AUTHOR>Mark Twain</AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>298</PAGES>
    <PRICE>$5.49</PRICE>
</BOOK>
<BOOK>
    <TITLE>Leaves of Grass</TITLE>
    <AUTHOR>Walt Whitman</AUTHOR>
    <BINDING>hardcover</BINDING>
    <PAGES>462</PAGES>
    <PRICE>$7.75</PRICE>
</BOOK>

```

Элементы также должны быть правильно вложены. При этом если элемент (ограниченный начальным и конечным тегами) начинается внутри другого элемента, то он должен и заканчиваться внутри того же элемента. Например, эти элементы являются корректно сформированными:

```

<BOOK>
    <TITLE>Leaves of Grass</TITLE>
    <AUTHOR>Walt Whitman</AUTHOR>
</BOOK>

```

В то же время эти элементы не являются корректно сформированными:

```
<BOOK><TITLE>The Adventures of Huckleberry Finn</BOOK></TITLE>
```

⇒ *Примечание.* Элемент, который содержит один или более вложенных элементов, называется родительским элементом. Элемент, содержащийся непосредственно внутри родительского элемента (например, TITLE внутри BOOK), называется дочерним элементом, субэлементом, или вложенным элементом.

3.2. АНАТОМИЯ ЭЛЕМЕНТА

Элемент обычно состоит из начального тега, содержимого и конечного тега (рис. 3.2).



Рис. 3.2. Содержание элемента

В отличие от HTML, XML требует наличия как начального, так и конечного тега. (Единственным исключением является элемент без содержимого, для которого вы можете использовать специальный тег пустого элемента, о котором пойдет речь далее в этой главе.)

Имя, которое содержится в стартовой позиции начального тега и в конечном теге (TITLE в нашем примере), называется типом, или родовым идентификатором элемента (GI). Имя элемента идентифицирует особый тип или класс элемента, а не собственно элемент. Таким образом, документ может содержать более одного элемента с одинаковыми именами типа (например, элементы BOOK или TITLE в листинге 3.1 на с. 43–44).

При добавлении элемента в XML-документ вы можете выбрать любое имя типа по вашему желанию, руководствуясь при этом следующими правилами:

1) имя должно начинаться с буквы или с символа подчеркивания (), после чего могут идти буквы, цифры, точки (.), тире (–) или символы подчеркивания;

2) в спецификации XML указано, что имена типов элементов, которые начинаются с префикса «xml» (при любом сочетании строчных или прописных букв), зарезервированы для стандартных имен. Хотя Internet Explorer 5 не слишком требователен в этом отношении, лучше не использовать этот префикс, дабы избежать будущих неприятностей.

Примеры правильного задания имен типов элементов:

- Part;
- _1stPlace;
- A;
- B-SECTION;
- Street.Address.1.

Следующие имена использовать недопустимо:

- 1) 1stPlace (в качестве первого символа нельзя использовать цифру);
- 2) B Section (пробел внутри имени не разрешается);
- 3) B/Section (косая черта внутри имени не допускается);
- 4) :Chapter (двоеточие нельзя использовать в качестве первого символа);
- 5) A:Section (в Internet Explorer 5 разрешается, только если вы объявили A как пространство имен).



Примечание. В соответствии с XML-спецификацией двоеточие (:) в имени элемента зарезервировано для задания пространства имен. Пространства имен дают возможность дифференцировать элементы с одними и теми же именами. Этот вопрос будет обсуждаться в 7.13 «Вставка элементов HTML в XML-документы и использование пространства имен». Internet Explorer 5 разрешает вам использовать двоеточие в имени элемента только в том случае, если оно следует за пространством имен, объявленным вами в документе. Например, запись A:Section будет корректной, только если вы объявили A как пространство имен.

Имя, записанное в начальном теге, должно в точности соответствовать имени в конечном теге, включая регистр, в котором набраны буквы. Так, следующий элемент является некорректным:

```
<Title>Chapter one</title>
```

Соблюдение регистра существенно для имен элементов, как и для всего текста в описании разметки. Так, тип элемента с именем Ace не будет эквивалентен типу ace или ACE.

3.2.1. Типы содержимого элемента

Содержимым элемента считается текст, расположенный между начальным и конечным тегами. Вы можете использовать в качестве содержимого элемента следующие типы сообщений:

– *вложенные элементы* – в листинге 3.1 (см. на с. 43–44) элементы INVENTORY и BOOK имеют в своем содержимом вложенные элементы (рис. 3.3);

Содержимое
элемента BOOK =
пять вложенных
элементов

```
<BOOK>
  <TITLE>The Scarlet Letter</TITLE>
  <AUTHOR>Nathaniel Hawthorne</AUTHOR>
  <BINDING>trade paperback</BINDING>
  <PAGES>253</PAGES>
  <PRICE>$4.25</PRICE>
</BOOK>
```

Рис. 3.3. Содержимое элемента BOOK

– *символьные данные* – это текст, выражающий информационное содержание элемента, например название определенной книги в элементе TITLE (рис. 3.4).

Содержимое элемента TITLE = символьные данные

```
<TITLE>The Scarlet Letter</TITLE>
```

Рис. 3.4. Содержимое элемента TITLE

Ниже приведен пример содержимого элемента, состоящего из сочетания символьных данных и вложенного элемента (рис. 3.5).

Символьные данные

Содержимое
элемента
TITLE

```
<TITLE>
  Moby-Dick
  <SUBTITLE>Or, the Whale</SUBTITLE>
</TITLE>
```

Вложенный элемент

Рис. 3.5. Содержимое элемента, состоящего из сочетания символьных данных и вложенного элемента

При добавлении в элемент символьных данных вы можете использовать любые символы, за исключением левой угловой скобки (<), амперсанда (&) и строки ([>).



Совет. Синтаксический анализатор XML сканирует символьные данные элемента для XML-разметки. Вы не можете использовать левую угловую скобку <, амперсанд & или строку]]> в составе символьных данных, поскольку анализатор может интерпретировать < как начало вложенного элемента, & как начало ссылки на примитив или символ, а]]> как окончание раздела CDATA. Если вы хотите использовать символы < или & как часть символьных данных, вам необходимо воспользоваться разделом CDATA. Вы также можете использовать любые символы (в том числе те, которых нет на клавиатуре), воспользовавшись ссылкой на символ. Некоторые символы (например, < или &) вы можете вставлять с использованием предопределенных ссылок на общие примитивы.

На рис. 3.6 приведен элемент, содержащий ссылки на общий примитив и символ.

```
<TITLEPAGE> |
Author: &author;
Document Name: "How to Enter the &#60; Character"
</TITLEPAGE>
```

Ссылка на символ

Рис. 3.6. Ссылки на общий примитив и символ

Пример раздела CDATA внутри элемента показан на рис. 3.7. Раздел CDATA – это текстовый блок, в котором вы можете свободно размещать любые символы, за исключением строки (]]>).

```
<TITLEPAGE>
  Author: Mike
  <![CDATA[
    Document Name: "How to Enter the < and & Characters"
  ]]>
</TITLEPAGE>
```

Раздел CDATA

Рис. 3.7. Пример раздела CDATA

Инструкции по обработке содержат информацию, необходимую для XML-приложений.

Комментарии – это примечания к вашему XML-документу, которые прочитываются людьми, но игнорируются XML-процессором.

На рис. 3.8 представлен элемент, содержащий инструкцию по обработке и комментарий.

Инструкция по обработке	—	<pre> <BOOK> <?MyApp Parm1="value 1" Parm2="value 2"?> <TITLE>The Legend of Sleepy Hollow</TITLE> <AUTHOR>Washington Irving</AUTHOR> </pre>
Комментарий	—	<pre> <!-- You can put a comment inside an element. --> <BINDING>mass market paperback</BINDING> <PAGES>98</PAGES> <PRICE>\$2.95</PRICE> </BOOK> </pre>

Рис. 3.8. Элемент, содержащий инструкцию по обработке и комментарий

Документ содержит следующие типы элементов:

- элемент с комментарием в качестве его содержимого (INVENTORY). Обратите внимание, что браузер не отображает текст комментария;

- пустой элемент с именем COVER_IMAGE в начале каждого элемента BOOK. Назначение этого элемента – указать XML-приложению отобразить определенную картинку для книжной обложки. (Атрибут Source содержит имя файла картинки.) Чтобы иметь возможность использовать такой элемент, вам потребуется отобразить XML-документ на HTML-странице посредством сценария либо с помощью XSL-таблицы стилей вместо того, чтобы использовать простую CSS-таблицу, как это делается в рассматриваемом примере;

- элемент (элемент TITLE для Moby-Dick), который содержит символьные данные и дочерний элемент (SUBTITLE). Заметим, что браузер отображает и символьные данные, и дочерний элемент в одной строке, и в одном и том же формате. (CSS-формат, назначенный элементу TITLE, наследуется элементом SUBTITLE.)

3.2.2. Пустые элементы

Вы также можете помещать пустой элемент – элемент, не имеющий содержимого, – в ваш документ. Пустой элемент создается путем размещения конечного тега сразу же после начального тега. Например:

```
<HR></HR>
```

Либо вы можете использовать специальный тег пустого элемента:

```
<HR/>
```

Обе эти нотации являются эквивалентными.

Поскольку пустой элемент не имеет содержимого, у вас может возникнуть вопрос о его предназначении. Здесь есть два варианта:

1) вы можете использовать пустой элемент, чтобы указать XML-приложению выполнить действие или отобразить объект. Аналогом в HTML является пустой элемент BR, который является указанием браузеру вставить разрыв строки, а также пустой элемент HR, указывающий на вставку горизонтальной разделительной линии. Другими словами, само присутствие элемента с определенным именем – без какого-либо содержимого – может послужить важной информацией для приложения;

2) пустой элемент может нести информацию посредством атрибутов, о которых вы узнаете далее в этой главе. (С элементами, которые имеют атрибуты, вы еще не встречались.) Аналогом в HTML является пустой элемент IMG (изображение), содержащий атрибуты, которые сообщают процессору, где искать графический файл и как его отобразить.



Примечание. Таблица каскадных стилей может использовать пустой элемент для отображения рисунка, подробнее об этом вы можете узнать в главе 7. В главе 9 вы научитесь использовать HTML-сценарии, а в главе 10 – XSL-таблицы стилей для доступа к элементам и их атрибутам и для последующего выполнения соответствующих действий.

3.3. ЗАДАНИЕ АТТРИБУТОВ ДЛЯ ЭЛЕМЕНТОВ

В начальный тег элемента либо в тег пустого элемента вы можете включить одно или несколько описаний атрибутов. Описание атрибута представляет собой пару «имя – значение», связанную с данным элементом. Например, следующий элемент PRICE включает атрибут с именем Type, которому присвоено значение retail:

```
<PRICE Type="retail">$10.95</PRICE>
```

Элемент BOOK содержит два атрибута – Category и Display:

```
<BOOK Category="fiction" Display="emphasize">
  <TITLE>The Marble Faun</TITLE>
  <AUTHOR>Nathaniel Hawthorne</AUTHOR>
  <BINDING>trade paperback</BINDING>
  <PAGES>473</PAGES>
  <PRICE>$10.95</PRICE>
</BOOK>
```

Следующий пустой элемент включает атрибут с именем Source, который указывает на имя файла, содержащего картинку, которую следует отобразить:

```
<COVER_IMAGE Source="Faun.gif"/>
```

Задание атрибутов обеспечивает альтернативный способ включения информации в элемент. Обычно вы помещаете все относящиеся к элементу данные, которые хотите отобразить, внутри содержимого элемента. Атрибуты же используются для хранения различных свойств элемента, которые не обязательно будут отображены (например, категория или указания по отображению). В спецификации XML не установлено строгих разграничений относительно типа информации, которую можно описывать с помощью атрибутов или внутри содержимого элемента.

⇒ *Примечание.* Когда вы отображаете XML-документ с использованием CSS-таблицы (данный метод будет обсуждаться в главе 7), браузер не выводит атрибуты или их значения. Отображение же XML-документа с использованием связывания данных, сценария для HTML-страницы либо XSL-таблицы стилей дает вам возможность иметь доступ к атрибутам и их значениям, а также отображать значение или выполнять соответствующие действия.

3.3.1. Правила для создания атрибутов

Описание атрибута состоит из имени атрибута, вслед за которым идет знак равенства и значение атрибута. Вы можете выбрать любое имя атрибута, придерживаясь при этом следующих правил:

- имя должно начинаться с буквы или символа подчеркивания (), после чего могут следовать или не следовать другие буквы, цифры, точки (.), тире (–) или символы подчеркивания;
- спецификация XML оговаривает, что имена атрибутов, начинающиеся с префикса «xml» (в любом сочетании строчных или прописных букв), зарезервированы для стандартного использования. Хотя для Internet Explorer 5 это ограничение значения не имеет, во избежание проблем в будущем лучше не использовать этот префикс; (каждое имя атрибута может только один раз присутствовать в одном и том же начальном теге или в теге пустого элемента).

Например, следующие описания имен в стартовых тегах являются допустимыми:

```
<ANIMATION FileName="Waldo.ani">
<LIST _1stPlace="Sam">
<ENTRY Zip.Code="94941">
```

Приведенные ниже имена атрибутов недопустимы:

```
<!-- Дублирование имени атрибута внутри одного тега. -->
<ANIMATION FileName="Waldo1.ani" Filename="Waldo2.ani">
<LIST 1stPlace=""Sam"> <!-- Первый символ не может
быть цифрой. -->
<ITEM A:Category="cookware">
```

3.3.2. Правила для корректного задания значений атрибутов

Значение, которое вы присваиваете атрибуту, представляет собой группу символов, ограниченных кавычками, называемую также литералом. Вы можете присвоить атрибуту в качестве значения любой литерал, придерживаясь при этом следующих правил:

1) строка может быть заключена как в одинарные ('), так и в двойные кавычки (");

2) строка не может содержать внутри себя тот же символ кавычек, которыми она ограничена;

3) строка может содержать ссылку на символ или ссылку на внутренние примитивы общего назначения (об этом будет рассказано в главе 6);

4) строка не может содержать символ < (синтаксический анализатор может воспринять этот символ как начало описания XML-разметки);

5) строка не может содержать символ &, если это не ссылка на символ или примитив.

Вы уже познакомились с правильными описаниями атрибутов. Ниже приведены неправильные описания:

– <EMPLOYEE Status= ""downsized""> (нельзя использовать символы-ограничители внутри строки);

– <ALBUM Type= "<CD>"> (нельзя применять символ < внутри строки);

– <WEATHER Forecast= "Cold & Windy"> (нельзя использовать символ &, если это не ссылка).

Если вы хотите использовать двойные кавычки (") внутри значения атрибута, вам следует применить в качестве ограничителей одинарные кавычки ('), как показано в примере:

```
<EMPLOYEE Status=' "downsized" '>
```

Аналогично, чтобы включить одинарные кавычки в значение атрибута, следует использовать в качестве ограничителей двойные кавычки:

```
<CANDIDATE name="W.T. 'Bill' Bagley">
```



Совет. Вы можете избавиться от всех ограничений и вводить в имя атрибута любой символ, если воспользуетесь ссылкой на символ или – если это возможно – ранее определенной ссылкой на примитив общего назначения. О таких ссылках будет рассказано более подробно в главе 6.

Если вы создаете корректно сформированный документ, не имеющий объявления типа документа (как вы делали в упражнениях в данной главе), вы можете присвоить атрибуту любое значение, соответствующее приведенным выше правилам. Однако если вы создаете описание типа документа и определяете внутри него атрибуты, то можете ограничить типы значений, которые могут быть присвоены конкретному атрибуту. Например, вы можете определить атрибут, которому могут быть присвоены только значения «yes» или «no». Таким образом, одно из преимуществ задания определенного типа информации через атрибуты элементов в отличие от задания через содержимое элемента заключается в том, что при этом вы можете обеспечить контроль надо всеми типами данных, которые могут быть присвоены атрибуту, и указываете синтаксическому анализатору учитывать эти ограничения типов. (Как вы узнаете из главы 5, в основной спецификации XML не приведены средства для ограничения типов символьных данных для элемента.)

ЗАДАНИЯ

1. Создайте различные типы элементов

Откройте новый, пустой текстовый файл в вашем текстовом редакторе и наберите XML-документ. Если хотите, можете воспользоваться документом `Inventory.xml`, созданным вами в главе 2, в качестве отправной точки.

Выберите команду `Save (Сохранить)` вашего текстового редактора, чтобы сохранить документ на жестком диске, присвоив ему имя `Inventory03.xml`.

⇒ *Примечание.* Созданный вами документ использует CSS с именем Inventory02.css, который вы создали в предыдущем упражнении. Проверьте, чтобы файл с этой таблицей стилей был в той же папке, что и файл Inventory03.xml.

В Windows Explorer (Проводник) или в окне папки дважды щелкните на имени ранее сохраненного файла Inventory03.xml. Internet Explorer 5 отобразит документ, как показано на рис. 3.9.

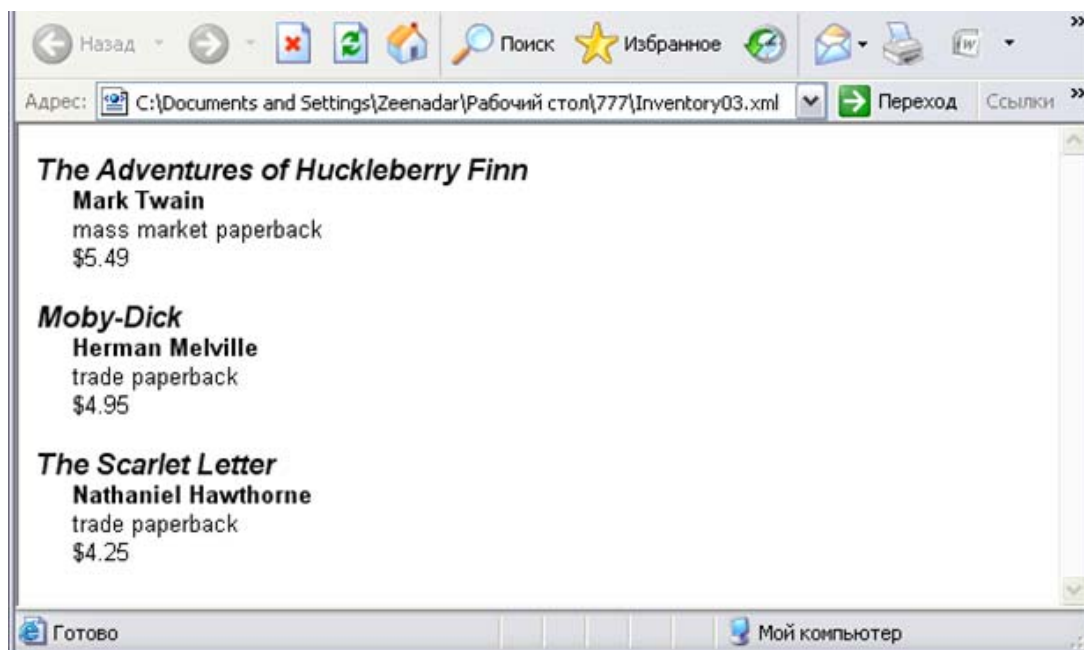


Рис. 3.9. Отображение в Internet Explorer 5
сохраненного файла Inventory03.xml

2. Преобразуйте содержимое в атрибуты

Откройте новый, пустой текстовый файл в вашем текстовом редакторе и наберите XML-документ. При желании можете воспользоваться ранее созданным вами документом Inventory.xml в качестве отправной точки.

Выберите команду Save (Сохранить) вашего текстового редактора, чтобы сохранить документ на жестком диске под именем Inventory04.xml.

⇒ *Примечание.* Созданный вами документ использует CSS-таблицу с именем Inventory02.css, которую вы создали в предыдущем упражнении. Проверьте, что этот файл таблицы стилей находится в той же папке, что и файл Inventory04.xml.

В Windows Explorer (Проводник) или в окне папки дважды щелкните на имени файла Inventory04.xml. Internet Explorer 5 отобразит документ, как показано на рис. 3.10.

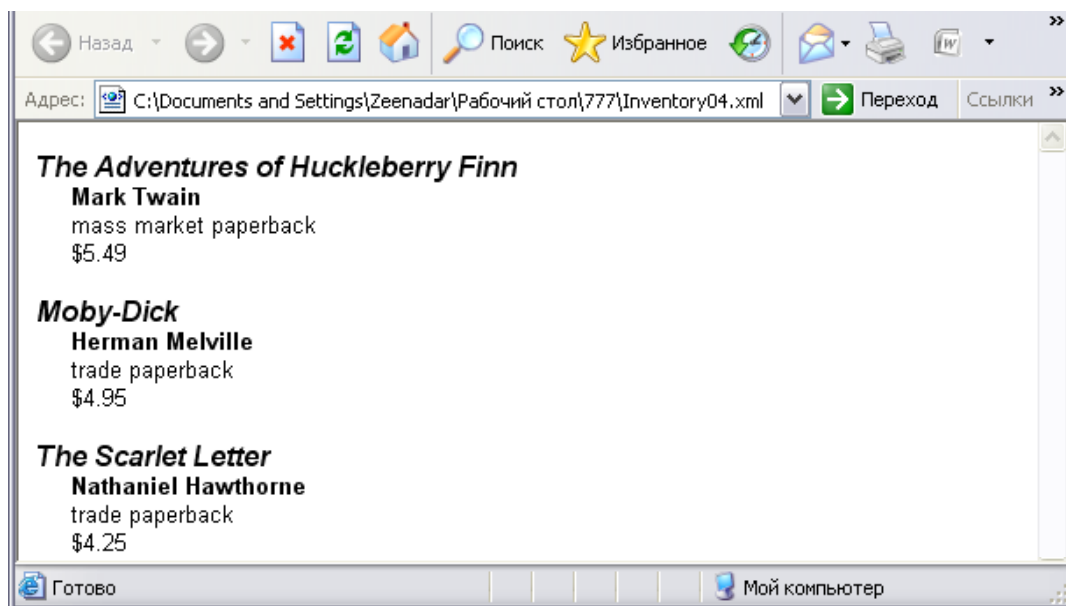


Рис. 3.10. Отображение в Internet Explorer 5 файла с использованием CSS-таблиц

Данный документ основан на документе Inventory.xml, который вы создали в одном из предыдущих упражнений, однако он содержит несколько дополнительных элементов. В частности, два внесенных изменения демонстрируют использование атрибутов:

- в каждом элементе BOOK информация о виде переплета преобразована из содержимого (в форме вложенного элемента BINDING) в атрибут с именем Binding. Это преобразование необходимо, если вы хотите хранить информацию о виде переплета, но не желаете ее показывать вместе с другой информацией о книге при представлении документа с использованием CSS-таблицы (Посмотрите на рис. 3.10 и убедитесь, что Internet Explorer 5 не отобразил значения атрибута.);

- к каждому элементу AUTHOR был добавлен атрибут с именем Born, содержащий дату рождения автора. Это пример малозначительной информации, которую вы хотели бы хранить, но отображать ее нет необходимости. Один из способов скрыть такую информацию и объявить ее малозначительной – назначить ее в качестве значения атрибуту, а не размещать в содержимом элемента.

Это только несколько из обширных возможностей применения атрибутов. Подробнее эти вопросы будут обсуждены в главе 5.

Глава 4. КОММЕНТАРИИ В XML. РАЗДЕЛЫ CDATA

4.1. ДОБАВЛЕНИЕ КОММЕНТАРИЕВ, ИНСТРУКЦИЙ ПО ОБРАБОТКЕ И РАЗДЕЛОВ CDATA

В этой главе вы узнаете, как добавлять и использовать в ваших документах три типа XML-разметки: комментарии, инструкции по обработке и разделы CDATA. Эти три составляющие не обязательны для корректно сформированных (или валидных) XML-документов, но они могут быть полезными. Вы можете использовать комментарии, чтобы сделать ваш документ более понятным для восприятия. С помощью инструкций по обработке вы можете изменить способ обработки или отображения ваших документов приложением. Вы также можете воспользоваться разделами CDATA, чтобы включать в символьные данные элемента практически любые сочетания символов.

4.2. ДОБАВЛЕНИЕ КОММЕНТАРИЕВ

Одним из принципов XML является создание предельно ясных и простых для понимания документов. Помещенные в нужном месте исчерпывающие примечания могут оказать существенную помощь при восприятии XML-документа подобно тому, как комментарии заметно облегчают чтение исходного кода программы на языке C++ или BASIC.



Примечание. В Microsoft Internet Explorer 5 XML-процессор не анализирует и не обрабатывает текст комментариев в XML-разметке. Тем не менее он делает тексты комментариев доступными для написанного внутри HTML-страницы кода сценария. В главе 9 вы узнаете, как применять сценарии для доступа к тексту комментария, а также к другим компонентам XML-документа. Internet Explorer 5 отображает все комментарии в XML-документе, если вы открыли документ в браузере и если документ не имеет соответствующей таблицы стилей.

4.2.1. Форма записи комментариев

Комментарий начинается с символов `<!--` и заканчивается символами `-->`. Между этими двумя ограничителями вы можете поместить любые символы, за исключением двойного тире (`--`). Вы даже можете вставлять внутрь комментария символ левой угловой скобки (`<`) и знак амперсанда (`&`). Вот пример правильно записанного комментария:

```
<!-- Здесь вы можете поместить любой текст, за исключением двойного тире. Символы < и & также допустимы! -->
```

4.2.2. Место для размещения комментария

Вы можете вставить комментарий в любое место в вашем XML-документе, но вне описания разметки, например поместить его в пролог документа.

Листинг 4.1. Комментарий, помещенный в пролог

```
<?xml version="1.0"?>
<!-- Это комментарий в прологе. -->
<DOCELEMENT>
This is a very simple XML-document.
</DOCELEMENT>
```

Вы можете разместить комментарий вслед за элементом Документ.

Листинг 4.2. Комментарий, размещенный за элементом Документ

```
<?xml version="1.0"?>
<DOCELEMENT>
This is a very simple XML-document.
</DOCELEMENT>
<!-- Это комментарий, следующий за элементом Документ. -->
```

Кроме того, вы можете вставить его внутри содержимого корневого элемента.

Листинг 4.3. Комментарий, являющийся частью содержимого корневого элемента

```
<?xml version="1.0"?>
<DOCELEMENT>
<!-- Это комментарий, который является частью содержимого корневого элемента. -->
This is a very simple XML-document.
</DOCELEMENT>
```

Ниже приведен пример неправильной записи комментария, поскольку он помещен внутри разметки.

Листинг 4.4. Неправильная запись комментария

```
<?xml version="1.0"?>
<DOCELEMENT <!-- Это неправильная запись комментария! --> >
This is a very simple XML-document.
</DOCELEMENT>
```

Тем не менее вы можете вставлять комментарий внутри определения типа элемента (DTD) – несмотря на то, что DTD является видом разметки, – если только при этом комментарий не находится внутри другой разметки, входящей в состав DTD. Подробнее о DTD и правилах размещения комментариев внутри него вы узнаете в главе 5.

4.3. ДОБАВЛЕНИЕ ИНСТРУКЦИЙ ПО ОБРАБОТКЕ

Назначение инструкций по обработке – сообщить информацию, передаваемую XML-процессором приложению.

⇒ *Примечание.* XML-процессор представляет собой программный модуль, который прочитывает и хранит содержимое XML-документа. Приложение – это отдельный программный модуль, который получает содержимое документа от XML-процессора, а затем обрабатывает и отображает это содержимое. Если вы отображаете XML-документ в Internet Explorer 5, браузер содержит как XML-процессор, так и часть приложения.

4.3.1. Форма записи инструкций по обработке

Инструкция по обработке имеет следующую общую форму записи:

```
<?Кому инструкция?>
```

Здесь «Кому» есть имя приложения, которому адресована инструкция. Допускается любое имя при соблюдении следующих правил:

- имя должно начинаться с буквы или символа подчеркивания (), после чего могут следовать или не следовать другие буквы, цифры, точки (.), тире (–) или символы подчеркивания ();

- имя «xml», в любом сочетании строчных или прописных букв, зарезервировано («xml» строчными буквами используется в объявлении XML-документа, которое представляет собой разновидность инструкции по обработке).

Инструкция есть информация, передаваемая приложению. Она может состоять из любой последовательности символов, за исключением пары ?>, зарезервированной для обозначения окончания инструкции по обработке.

4.3.2. Особенности использования инструкций по обработке

В зависимости от процессора, который будет прочитывать документ, вы можете применять различные инструкции по обработке. Если вы используете в качестве XML-процессора Internet Explorer 5, у вас есть две основные возможности применения инструкций по обработке:

1) вы можете использовать стандартные, зарезервированные инструкции, чтобы сообщить Internet Explorer 5, как отображать документ с применением соответствующей таблицы стилей. Например, следующая инструкция по обработке предписывает Internet Explorer 5 использовать CSS-таблицу из файла Inventory01.css:

```
<?xml-stylesheet type="text/css" href=" Inventory01.css"?>
```

2) если вы пишете Web-сценарий для управления и отображения XML-документа, вы можете поместить в документ любые незарезервированные инструкции по обработке, а ваша программа-сценарий будет считывать эти инструкции и выполнять определенные действия. Например, вы можете вставить в документ следующую инструкцию по обработке, сообщающую вашему сценарию уровень детализации при отображении:

```
<?MyScript detail="2"?>
```

В главе 9 вы узнаете, как использовать сценарии для доступа к составляющим XML-документа, включая инструкции по обработке.

4.3.3. Место для размещения инструкций по обработке

Вы можете поместить инструкцию по обработке в любое место XML-документа вне других элементов разметки, т. е. вы можете помещать их аналогично комментариям: в пролог документа, после элемента Документ либо внутри содержимого элемента. Ниже при-

веден XML-документ с правильно записанными инструкциями по обработке.

Листинг 4.5. Правильно записанные инструкции по обработке XML-документа

```
<?xml version="1.0"?>
<!-- Далее следует инструкция по обработке внутри пролога: -->
<?xml-stylesheet type="text/css" href="Inventory01.css"?>
<INVENTORY>
  <BOOK>
    <!-- Инструкция по обработке внутри содержимого
    элемента: -->
    <?ScriptA emphasize="yes"?>
    <TITLE> The Adventures of Huckleberry Finn</TITLE>
    <AUTHOR>Mark Twain</AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>298</PAGES>
    <PRICE>$5.49</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>Leaves of Grass</TITLE>
    <AUTHOR>Walt Whitman</AUTHOR>
    <BINDING>hardcover</BINDING>
    <PAGES>462</PAGES>
    <PRICE>$7.75</PRICE>
  </BOOK>
</INVENTORY>
<!-- Инструкция по обработке после элемента Документ: -->
<?ScriptA Category="books" Style="formal" ?>
```

Вот пример инструкции по обработке, неверно помещенной внутри элемента разметки.

Листинг 4.6. Неверно записанная инструкция по обработке XML-документа

```
<BOOK <?ScriptA emphasize="yes"?> >
  <TITLE>Leaves of Grass</TITLE>
  <AUTHOR>Walt Whitman</AUTHOR>
  <BINDING>hardcover</BINDING>
  <PAGES>462</PAGES>
  <PRICE>$7.75</PRICE>
</BOOK>
```

Тем не менее вы можете помещать инструкцию по обработке внутри определения типа документа (DTD) – несмотря на то, что DTD представляет собой форму разметки, – если только она не находится внутри имеющегося в DTD другого элемента разметки.

4.4. ДОБАВЛЕНИЕ РАЗДЕЛОВ CDATA

Внутри символьных данных в содержимом элемента нельзя помещать символ левой угловой скобки (<) или знак амперсанда (&). Одним из способов преодолеть это ограничение является использование ссылки на символ (< или &) либо на предопределенный общий примитив (< или &). Однако в том случае, если вам требуется многократно вставлять символы < или &, применение ссылок неудобно и затрудняет восприятие данных. В этом случае проще поместить текст, содержащий такие символы, в раздел CDATA.

4.4.1. Форма записи раздела CDATA

Раздел CDATA начинается с символов <![CDATA[и заканчивается символами]]>. Между этими двумя ограничителями вы можете поместить любые символы (включая < или &), за исключением]]> (что будет интерпретировано как конец раздела CDATA). Все символы внутри раздела CDATA трактуются как литеральная часть символьных данных элемента, а не как XML-разметка.

Ниже приведен пример правильно записанного раздела CDATA:

```
<![CDATA[ Здесь вы можете разместить любые символы, за исключением двух правых квадратных скобок с последующим знаком "больше". ]]>
```



Примечание. Ключевое слово CDATA (как и другие ключевые слова XML) должно быть набрано прописными буквами.

Если вы хотите включить в состав имеющихся символьных данных блок исходного кода или разметку, которые будут отображаться браузером, то можете воспользоваться разделом CDATA с целью предотвратить интерпретацию синтаксическим анализатором символов < или & как XML-разметку. Например:

```
<A-SECTION>
```

Вот пример очень простой HTML-страницы.

Листинг 4.7. Пример использования раздела CDATA

```
<![CDATA[  
<HTML>  
<HEAD>
```

```

<TITLE>R. Jones & Sons</TITLE>
</HEAD>
<BODY>
<P>Добро пожаловать на нашу домашнюю страницу!</P>
</BODY>
</HTML>
]]>
</A-SECTION>

```

Внутри раздела CDATA процессор будет предполагать, что <HTML>, например, есть начало вложенного элемента, но не часть символьных данных элемента A-SECTION.

⇒ *Примечание.* Поскольку вы можете непосредственно помещать символы < и & внутрь раздела CDATA, вам не нужно использовать ссылки на символы (< и &) либо на predefined примитивы общего назначения (< и &). Фактически, если вы используете подобную ссылку, синтаксический анализатор интерпретирует каждый из символов в ссылке как литерал и не замещает ссылку символами < и &.

4.4.2. Место для размещения раздела CDATA

Вы можете вставить раздел CDATA в любое место, занимаемое символьными данными, т. е. внутри содержимого элемента, но не внутри XML-разметки. Вот правильно записанный раздел CDATA.

Листинг 4.8. Правильная запись раздела CDATA

```

<?xml version="1.0"?>
<MUSICAL>
  <TITLE_PAGE>
    <![CDATA[
      <Oklahoma!>
        By
        Rogers & Hammerstein
    ]]>
  </TITLE_PAGE>
  <!-- Здесь расположены другие элементы... -->
</MUSICAL>

```

Ошибочно сформированный XML-документ, представленный ниже, содержит два неправильно записанных раздела CDATA. Первый из них не находится внутри содержимого элемента. Вторым

помещен внутри содержимого элемента, но также и внутри начального тега разметки.

Листинг 4.9. Неправильная запись разделов CDATA

```
<?xml version="1.0"?>
<![CDATA[ ОШИБКА: не внутри содержимого элемента! ]]>
<DOC_ELEMENT>
  <SUB_ELEMENT <![CDATA[ ОШИБКА: внутри разметки! ]]> >
    содержимое подэлемента...
  </SUB_ELEMENT>
</DOC_ELEMENT>
```

⇒ *Примечание.* Разделы CDATA не являются вложениями. Вы не можете поместить один раздел CDATA внутрь другого.

Глава 5. ВАЛИДНОСТЬ XML-ДОКУМЕНТОВ. ПОДМНОЖЕСТВА DTD

5.1. СОЗДАНИЕ ВАЛИДНЫХ XML-ДОКУМЕНТОВ

Валидные XML-документы отвечают более строгому набору критериев, чем обычные корректно сформированные документы, с которыми вы имели дело в предыдущих главах. В этой главе вы прежде всего познакомитесь с основными требованиями, предъявляемыми к валидным XML-документам, оцените преимущества валидных документов. Затем вы узнаете, как создавать объявление типа документа, которое требуется для всех валидных XML-документов. После этого будут представлены подробные инструкции по определению элементов и атрибутов в валидном документе.

5.1.1. Основной критерий для валидного документа

Каждый XML-документ должен быть корректно сформированным, т. е. отвечать минимальным требованиям по составлению XML-документа. Если документ не является корректно сформированным, он не может считаться XML-документом.

Корректно сформированный XML-документ также может быть валидным. Валидным (valid) называется корректно сформированный (well-formed) документ, отвечающий двум дополнительным требованиям:

- пролог документа должен содержать специальное объявление типа документа, которое содержит определение типа документа (DTD), задающее структуру документа;
- остальной документ должен отвечать структуре, заданной в DTD.

В последующих подглавах этой главы, а также в главе 6 вы узнаете, как создавать документы, отвечающие этим двум общим требованиям.

5.1.2. Требования корректности формирования и валидности

Требования корректности формирования представляют собой набор правил, которые определены в спецификации XML и которым вы должны следовать (в дополнение к основным синтаксическим

требованиям), чтобы создать правильно составленный документ. Поскольку XML-документ должен быть корректно сформированным, любое отклонение от требований корректности формирования считается фатальной ошибкой (fatal error). Если XML-процессор сталкивается с фатальной ошибкой, он должен остановить нормальную обработку документа и не пытаться ее возобновить.

Требования валидности представляют собой дополнительный набор правил в спецификации XML, которым вы должны следовать, чтобы создать валидный документ. Поскольку валидность является не обязательной для XML-документа, отклонение от требований валидности считается лишь ошибкой (error), но не фатальным сбоем. Если XML-процессор встречает ошибку, он может просто выдать сообщение о ней и продолжить выполнение обработки. Требования валидности состоят из специальных правил по созданию соответствующего объявления типа с его DTD, а также созданию документа, отвечающего описанию внутри вашего DTD.

5.1.3. Преимущества использования валидных XML-документов

Может показаться, что при создании валидного XML-документа выполняется ряд лишних действий: вы должны сначала полностью описать структуру документа в DTD, а затем создать сам документ, отвечающий всем спецификациям, которые содержатся в DTD.

Однако если вы хотите быть уверенным, что ваш документ отвечает определенной структуре или набору стандартов, включение DTD, которое описывает эту структуру, дает возможность XML-процессору проверить, соответствует ли документ структуре. Другими словами, DTD обеспечивает стандартный шаблон для процессора, чтобы при проверке валидности он мог следовать требуемой структуре и гарантировать, что ваш документ соответствует установленным стандартам. Если какая-либо часть документа не отвечает DTD-спецификации, процессор может отобразить сообщение об ошибке.

Использование валидных документов особенно полезно для проверки однородности среди группы схожих документов. Фактически, стандарт XML определяет DTD как «грамматику для определенного класса документов».

Предположим, компании, занимающейся публикацией в Web, требуется, чтобы все ее редакторы создавали XML-документы,

отвечающие единой структуре. Создание одного DTD и включение его во все документы может обеспечить условия соответствия документов требуемой структуре, в то время как редакторы будут избавлены от необходимости добавлять новые элементы, помещать информацию в неправильном порядке, присваивать неправильные типы данных и т. д. Конечно, валидность документа должна быть проверена при его обработке процессором.

Включение DTD и проверка валидности имеют особое значение, если документ будет обрабатываться программой пользователя, ориентированной на определенную структуру документа. Если все пользователи программного обеспечения включают в свои XML-документы соответствующие DTD и все документы пройдут проверку на валидность, то пользователи могут быть уверены, что их документы будут распознаны программой-обработчиком. Например, если группа математиков создает математические документы, которые будут обрабатываться специальной программой, все они могут включить в свои документы одинаковые DTD, которые содержат определения требуемой структуры, элементов, атрибутов и других компонентов.

На деле большинство реальных XML-приложений, список которых приведен в конце главы 1, например MathML, состоят из стандартного DTD, которое все пользователи приложения включают в свои XML-документы, чтобы при проверке валидности обеспечивалось соответствие структуре приложения и документы были распознаны любой программой, разработанной для этого приложения.

Чтобы проверить документ на валидность, вы можете использовать сценарий проверки на валидность, приведенный в главе 9 в пункте «Проверка валидности XML-документа» (см. на с. 291). Вы можете прочесть приведенные там указания сейчас, чтобы иметь возможность осуществлять проверку на валидность создаваемых вами XML-документов.

5.2. ФОРМА ЗАПИСИ DTD

Объявление типа документа имеет следующую обобщенную форму записи:

```
<!DOCTYPE Имя DTD>
```

Здесь Имя указывает на имя элемента Документ. Имя действительного элемента Документ должно в точности соответствовать

имени, записанному в объявлении. Например, если вы создаете объявление типа документа для документа, рассмотренного в предыдущей главе, вам следует использовать имя INVENTORY:

```
<!DOCTYPE INVENTORY DTD>
```

DTD представляет собой определение типа документа, которое содержит объявления, задающие элементы документа, атрибуты и другие компоненты. В следующей подглаве вы познакомитесь с формой их записи.

⇒ *Примечание.* Подобно всем ключевым словам XML, DOCTYPE должно быть набрано прописными буквами.

5.2.1. Создание DTD

DTD состоит из символа левой квадратной скобки ([), после которой следует ряд объявлений разметки, заканчивающихся правой квадратной скобкой (]). Объявления разметки описывают логическую структуру документа, т. е. задают элементы документа, атрибуты и другие компоненты. На рис. 5.1 приведен законченный валидный XML-документ, содержащий DTD с единственным объявлением разметки.

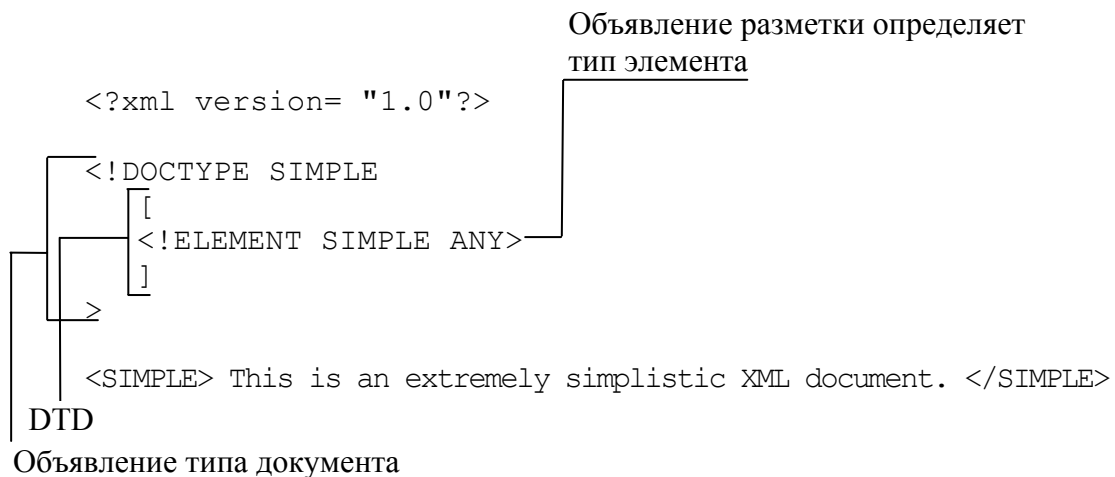


Рис. 5.1. Законченный валидный XML-документ, содержащий DTD с единственным объявлением разметки

DTD в этом примере документа указывает, что документ может содержать только элементы типа SIMPLE (это единственный заданный тип элемента) и что элемент SIMPLE может иметь

любое допустимое для данного типа содержимое (ключевое слово ANY).

DTD может содержать следующие типы объявлений разметки:

1) объявления типов элементов. Они определяют типы элементов, которые может содержать документ, а также содержимое и порядок следования элементов. Об объявлениях типов элементов пойдет речь в следующей подглаве;


2) объявления списков атрибутов. Каждое объявление списков атрибутов задает имена атрибутов, которые могут быть использованы с определенным типом элемента, а также типы данных и устанавливаемые по умолчанию значения этих атрибутов. Эти объявления будут рассмотрены далее в этой главе;

3) объявления примитивов. Вы можете использовать примитивы для хранения часто используемых фрагментов текста или для встраивания не относящихся к XML данных в ваш документ. Об объявлениях примитивов будет рассказано в главе 6;

4) объявления нотаций. Нотация описывает формат данных или идентифицирует программу, используемую для обработки определенного формата;

5) инструкции по обработке. Эта тема затрагивалась в 4.3 «Использование инструкций по обработке»;

6) ссылки на параметрические примитивы. Любой из приведенных выше компонентов может содержаться внутри параметрического примитива и добавляться путем ссылки на параметрический примитив. Смысл этого оператора станет вам понятен, когда вы познакомитесь с материалом из главы 6. Данный пункт приведен здесь для полноты картины.

 *Примечание.* Типы DTD, обсуждаемые в этой подглаве (и используемые в примерах в последующих подглавах), относятся к внутреннему подмножеству DTD, поскольку полностью включаются в объявление типа данного документа. В конце этой главы вы узнаете, как использовать DTD, размещенные в отдельном файле и относящиеся к внешнему подмножеству DTD.

5.2.2. Добавление DTD

Объявление типа документа представляет собой блок XML-разметки, который вы должны добавить в пролог валидного XML-документа. Он может располагаться в любом месте пролога – вне другой разметки – после XML-объявления, как показано на рис. 5.2.

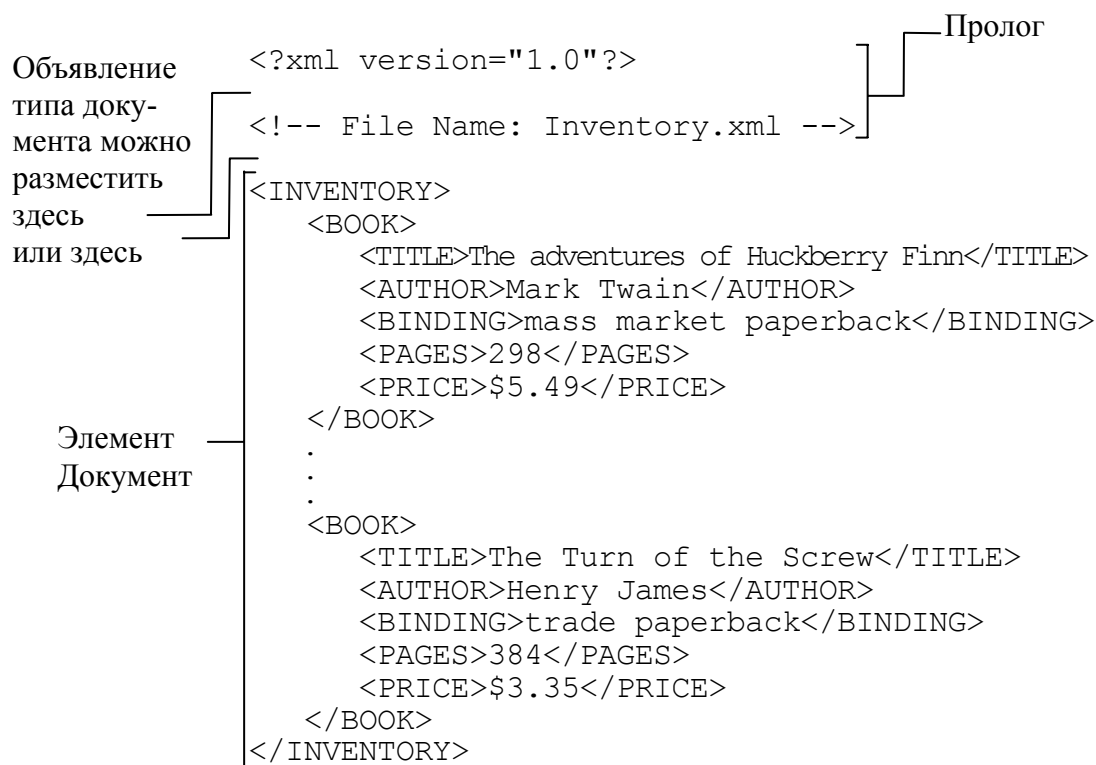


Рис. 5.2. Расположение блока XML-разметки

Объявление типа документа определяет структуру документа. Если вы открываете документ, не содержащий объявления типа, в Internet Explorer 5, процессор Internet Explorer 5 всего лишь осуществляет проверку документа на корректность формы составления. Если же вы открываете документ, содержащий объявление типа документа, процессор Internet Explorer 5 будет проверять документ на валидность вместе с проверкой на корректность формы составления, так что ваш документ должен отвечать всем имеющимся декларациям в объявлении типа документа. Так, вы не сможете включить в документ какие-либо элементы или атрибуты, если вы не указали их в объявлении типа документа. Каждый элемент и атрибут, который вы включаете, должен соответствовать спецификации, выраженной в соответствующем объявлении.

5.3. ОБЪЯВЛЕНИЕ ТИПОВ ЭЛЕМЕНТОВ

В валидном XML-документе вы должны полностью объявить тип каждого элемента, который вы используете в документе, в объявлении типа элемента внутри DTD. Объявление типа элемента указывает на имя типа элемента и допустимое содержимое элемента.

Как единое целое, объявление типа элемента в DTD – подобно построению базы данных – задает полную логическую структуру документа. Таким образом, объявление типа элемента указывает на типы элементов, которые содержит документ, порядок следования элементов, а также описание содержимого элементов.

Форма записи объявления типа элемента

Объявление типа элемента имеет следующую обобщенную форму:

```
<!ELEMENT Имя опись_содержимого>
```

Здесь Имя есть имя объявляемого типа элемента. Опись_содержимого – это описание содержимого, которое определяет, что может содержать элемент.

Ниже приведено объявление типа элемента с именем TITLE, для содержимого которого могут использоваться только символьные данные (дочерние элементы не допускаются):

```
<!ELEMENT TITLE (#PCDATA)>
```

А вот объявление для типа элемента с именем GENERAL, содержимое которого может быть любым:

```
<!ELEMENT GENERAL ANY>
```

В качестве последнего примера рассмотрим законченный XML-документ с двумя типами элементов. Объявление типа элемента COLLECTION указывает, что он может содержать один или несколько элементов CD, а объявление типа элемента CD свидетельствует о том, что он может содержать только символьные данные. Заметим, что документ соответствует этим объявлениям и, следовательно, является валидным.

Листинг 5.1. Тип элемента COLLECTION

```
<?xml version="1.0"?>
<!DOCTYPE COLLECTION
[
  <!ELEMENT COLLECTION (CD)+>
  <!ELEMENT CD (#PCDATA)>
  <!-- Вы также можете включать комментарии в DTD. -->
]
>
```

```
<COLLECTION>
  <CD>Mozart Violin Concertos 1, 2, and 3</CD>
  <CD>Telemann Trumpet Concertos</CD>
  <CD>Handel Concerti Grossi Op. 3</CD>
</COLLECTION>
```

⇒ *Примечание.* Вы можете объявить определенный тип элемента в данном документе только один раз.

5.4. ОПИСАНИЕ СОДЕРЖИМОГО ЭЛЕМЕНТА

Вы можете описать содержимое элемента, т. е. заполнить часть описи_содержимого в объявлении типа элемента, четырьмя различными способами:

– пустое содержимое (EMPTY). Ключевое слово EMPTY указывает, что элемент должен быть пустым, т. е. не может иметь содержимого. Например:

```
<!ELEMENT IMAGE EMPTY>
```

Ниже приведены валидные элементы IMAGE, которые вы можете поместить в документ:

```
<IMAGE></IMAGE>
<IMAGE/>
```

– любое содержимое (ANY). Ключевое слово ANY говорит о том, что элемент может иметь любое допустимое для этого типа содержимое. Иными словами, элемент этого типа может содержать или не содержать дочерние элементы в любом порядке и с любым количеством вхождений, иметь или не иметь чередующиеся символьные данные. Это наиболее неопределенный тип описания содержимого и дает возможность создавать типы элементов без ограничений на их содержимое. Вот пример соответствующего объявления:

```
<!ELEMENT MISC ANY>
```

– содержимое элемента (также называемое «дочернее содержимое»). При таком описании типа содержимого элемент может содержать дочерние элементы, но не может непосредственно содержать символьные данные. Об этой возможности будет говориться в следующей подглаве;

– смешанное содержимое. При этом описании типа содержимого элемент может содержать любое количество смешанных данных, в том числе и чередующихся с дочерними элементами определенных типов. Эта возможность будет рассмотрена далее в этой главе.

5.4.1. Задание содержимого элемента

Если элемент имеет содержимое, он может непосредственно содержать только определенные дочерние элементы, но не символьные данные. В тексте документа вы можете разделять дочерние элементы пробелами, чтобы улучшить восприятие документа, но процессор будет игнорировать символы пробела и не передаст их приложению.

Рассмотрим следующий пример XML-документа, описывающий одну книгу.

Листинг 5.2. Пример описания дочерних элементов

```
<?xml version="1.0"?>
<!DOCTYPE BOOK
[
  <!ELEMENT BOOK (TITLE, AUTHOR)>
  <!ELEMENT TITLE (#PCDATA)>
  <!ELEMENT AUTHOR (#PCDATA)>
]>
<BOOK>
  <TITLE>The Scarlet Letter</TITLE>
  <AUTHOR>Nathaniel Hawthorne</AUTHOR>
</BOOK>
```

На рис. 5.3 показано, как это будет отображено в браузере.

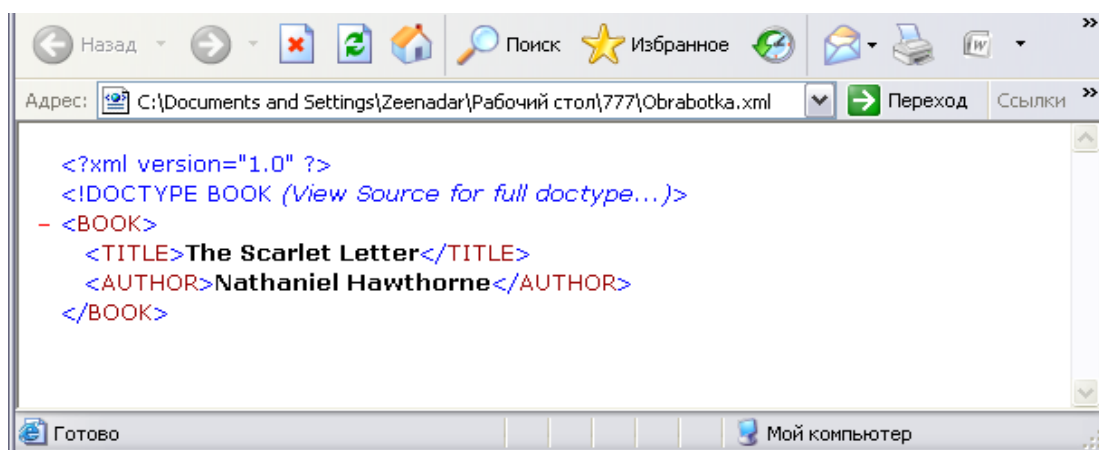


Рис. 5.3. Отображение в Internet Explorer 5 дочерних элементов

В данном документе тип элемента BOOK объявлен как имеющий содержимое элемента. TITLE, AUTHOR, следующие за именем элемента в объявлении, составляют модель содержимого. Модель содержимого указывает на разрешенные типы дочерних элементов и их порядок. В этом примере модель содержимого отображает то, что элемент BOOK должен иметь ровно один дочерний элемент TITLE, за которым следует ровно один дочерний элемент AUTHOR. При обработке документа процессор игнорирует три пустых строки, используемые для разделения дочерних элементов внутри элемента BOOK.

5.4.2. Модель содержимого

Модель содержимого может иметь одну из следующих основных форм:

1) последовательная. Последовательная форма модели содержимого указывает, что элемент должен иметь заданную последовательность дочерних элементов. При этом вы отделяете имена типов дочерних элементов запятыми. Например, следующее DTD указывает, что элемент MOUNTAIN должен иметь один дочерний элемент NAME, после которого идет один дочерний элемент HEIGHT, за которым следует один дочерний элемент STATE:

```
<!DOCTYPE MOUNTAIN
[
  <!ELEMENT MOUNTAIN (NAME, HEIGHT, STATE)>
  <!ELEMENT NAME (#PCDATA)>
  <!ELEMENT HEIGHT (#PCDATA)>
  <!ELEMENT STATE (#PCDATA)>
]>
```

Следовательно, следующий элемент Документ будет валидным:

```
<MOUNTAIN>
  <NAME>Wheeler</NAME>
  <HEIGHT>13161</HEIGHT>
  <STATE>New Mexico</STATE>
</MOUNTAIN>
```

Нижеприведенный элемент Документ, однако, не будет валидным, поскольку порядок дочерних элементов не соответствует объявленному:

```
<MOUNTAIN>  <!-- Неправильный элемент! -->
  <STATE>New Mexico</STATE>
  <NAME>Wheeler</NAME>
  <HEIGHT>13161</HEIGHT>
</MOUNTAIN>
```

Пропуск дочернего элемента или использование одного и того же типа дочернего элемента более одного раза также недопустимо. Как вы видите, это достаточно строгий вид объявления;

2) выборочная. Выборочная форма модели содержимого свидетельствует о том, что элемент может иметь любой из серии допустимых дочерних элементов, разделяемых символом |. Например, следующее DTD указывает, что элемент FILM может состоять из одного дочернего элемента STAR, или одного дочернего элемента NARRATOR, или одного дочернего элемента INSTRUCTOR:

```
<!DOCTYPE FILM
[
  <!ELEMENT FILM (STAR | NARRATOR | INSTRUCTOR)>
  <!ELEMENT STAR (#PCDATA)>
  <!ELEMENT NARRATOR (#PCDATA)>
  <!ELEMENT INSTRUCTOR (#PCDATA)>
]>
```

Следовательно, следующий элемент Документ будет валидным:

```
<FILM>
  <STAR>Robert Redford</STAR>
</FILM>
```

как и элемент:

```
<FILM>
  <NARRATOR>Sir Gregory Parsloe</NARRATOR>
</FILM>
```

а также элемент:

```
<FILM>
  <INSTRUCTOR>Galahad Threepwood</INSTRUCTOR>
</FILM>
```

Нижеприведенный элемент Документ не будет валидным, поскольку вы можете включить только один из дочерних элементов:

```
<FILM>  <!-- Неправильный элемент! -->
  <NARRATOR>Sir Gregory Parsloe</NARRATOR>
  <INSTRUCTOR>Galahad Threepwood</INSTRUCTOR>
</FILM>
```

Вы можете изменить любую из этих форм модели содержимого, используя символы: знак вопроса (?), знак плюс (+) и звездочку (*), значения которых описаны в таблице.

Значение символов дочерних элементов

Символ	Значение
?	Ни одного или один из предшествующих элементов
+	Один или более из предшествующих элементов
*	Ни одного или более из предшествующих элементов

Например, следующее объявление означает, что вы можете включить один или более дочерних элементов NAME и что дочерний элемент HEIGHT является необязательным:

```
<!ELEMENT MOUNTAIN (NAME+, HEIGHT?, STATE)>
```

Таким образом, следующий элемент будет правильным:

```
<MOUNTAIN>
  <NAME>Pueblo Peak</NAME>
  <NAME>Taos Mountain</NAME>
  <STATE>New Mexico</STATE>
</MOUNTAIN>
```

Другой пример: следующее объявление означает, что вы можете включить несколько или ни одного дочернего элемента STAR, либо один дочерний элемент NARRATOR, либо один дочерний элемент INSTRUCTOR:

```
<!ELEMENT FILM (STAR* | NARRATOR | INSTRUCTOR)>
```

Соответственно, каждый из следующих трех элементов будет корректным:

```
<FILM>
  <STAR>Tom Hanks</STAR>
  <STAR>Meg Ryan</STAR>
</FILM>
<FILM>
  <NARRATOR>Sir Gregory Parsloe</Narrator>
</FILM>
<FILM/>
```

Вы также можете воспользоваться символами ?, + или * для модификации всей модели содержимого, помещая символы непосредственно после закрывающих скобок. Например, следующее объявление дает вам возможность включить один или несколько дочерних элементов любого из этих трех типов в любом порядке:

```
<!ELEMENT FILM (STAR | NARRATOR | INSTRUCTOR)+>
```

Такое объявление делает корректными следующие элементы:

```
<FILM>
  <NARRATOR>Bertram Wooster</NARRATOR>
  <STAR>Sean Connery</STAR>
  <NARRATOR>Plug Basham</NARRATOR>
</FILM>
<FILM>
  <STAR>Sean Connery</STAR>
  <STAR>Meg Ryan</STAR>
</FILM>
<FILM>
  <INSTRUCTOR>Stinker Pike</INSTRUCTOR>
</FILM>
```

Наконец, вы можете формировать более сложные модели содержимого путем вложения выборочной модели содержимого внутрь последовательной модели, либо последовательной модели в выборочную модель. Например, следующее DTD задает, что каждый элемент FILM должен иметь один дочерний элемент TITLE; за ним должен следовать один дочерний элемент CLASS; после него должен идти один дочерний элемент STAR, NARRATOR или INSTRUCTOR:

```
<!DOCTYPE FILM
[
  <!ELEMENT FILM (TITLE, CLASS, (STAR | NARRATOR |
    INSTRUCTOR) )>
  <!ELEMENT TITLE (#PCDATA)>
  <!ELEMENT CLASS (#PCDATA)>
  <!ELEMENT STAR (#PCDATA)>
  <!ELEMENT NARRATOR (#PCDATA)>
  <!ELEMENT INSTRUCTOR (#PCDATA)>
]>
```

В соответствии с этим DTD следующий элемент Документ будет корректным:

```
<FILM>
  <TITLE>The Net</TITLE>
  <CLASS>fictional</CLASS>
  <STAR>Sandra Bullok</STAR>
</FILM>
```

так же, как такой:

```
<FILM>
  <TITLE>How to Use XML</TITLE>
  <CLASS>instructional</CLASS>
  <INSTRUCTOR>Penny Donaldson</INSTRUCTOR>
</FILM>
```

5.5. ЗАДАНИЕ СМЕШАННОГО СОДЕРЖИМОГО

Если элемент имеет смешанное содержимое, он может включать символьные данные. Если же вы зададите в объявлении один или несколько типов дочерних элементов, он может содержать любые из этих дочерних элементов в любом порядке и с любым количеством вхождений (нуль и более). Другими словами, при смешанном содержимом вы можете определять типы дочерних элементов, но не можете задавать порядок или количество вхождений дочерних элементов, а также обязательность включения для определенных типов дочерних элементов.

Чтобы объявить тип элемента смешанного содержимого, вы можете воспользоваться одной из следующих форм модели содержимого:

- только символьные данные. Чтобы объявить тип элемента, который может содержать только символьные данные, используйте модель содержимого (`#PCDATA`). Так, следующее объявление указывает, что для элемента `SUBTITLE` допускаются только символьные данные:

```
<!ELEMENT SUBTITLE (#PCDATA)>
```

Следующие два элемента, согласно данной декларации, являются корректными:

```
<SUBTITLE>A New Approach</SUBTITLE>
<SUBTITLE></SUBTITLE>
```

Заметим, что элемент, который в соответствии с объявлением должен содержать символьные данные, может и не иметь никаких символов, т. е. быть пустым.



Примечание. Ключевое слово PCDATA относится к синтаксически анализируемым (разбираемым) символьным данным. Из главы 3 вам известно, что XML-процессор синтаксически разбирает символьные данные внутри элемента, т. е. сканирует элемент в поиске XML-разметки. В связи с этим вы не можете использовать левую угловую скобку (<), или знак амперсанда (&), или символы]]> как часть символьных данных, поскольку синтаксический анализатор будет интерпретировать каждый из этих символов или группы символов как разметку. Однако вы можете использовать любые символы с помощью ссылки на символ или на предопределенный примитив, либо с помощью раздела CDATA.

– символьные данные с необязательными дочерними элементами. Чтобы объявить тип элемента, который может содержать символьные данные плюс ни одного или несколько дочерних элементов, перечислите каждый тип дочернего элемента после ключевого слова PCDATA в модели содержимого, разделяя их символами (и помещая звездочку (*) в конце всей модели содержимого. Каждое имя элемента может появляться в модели содержимого только один раз. Например, следующее объявление указывает, что элемент TITLE может содержать символьные данные плюс ни одного или несколько дочерних элементов SUBTITLE:

```
<!ELEMENT TITLE (#PCDATA | SUBTITLE)*>
```

В соответствии с этим объявлением следующие элементы TITLE являются допустимыми:

```
<TITLE>
  Moby-Dick
  <SUBTITLE>Or, the Whale</SUBTITLE>
</TITLE>
<TITLE>
  <SUBTITLE>Or, the Whale</SUBTITLE>
  Moby-Dick
</TITLE>
<TITLE>
  Moby-Dick
</TITLE>
<TITLE>
  <SUBTITLE>Or, the Whale</SUBTITLE>
  <SUBTITLE>Another Subtitle</SUBTITLE>
</TITLE>
```

5.6. ОБЪЯВЛЕНИЕ АТТРИБУТОВ

В валидном XML-документе вы также должны исчерпывающе объявить все атрибуты, которые вы предполагаете использовать для элементов документа. Вы определяете все атрибуты, ассоциированные с определенным элементом, с помощью специального типа DTD-разметки, называемого объявлением списка атрибутов. Это объявление:

1) определяет имена атрибутов, ассоциированных с элементом. В валидном документе вы можете включить в начальный тег элемента только те атрибуты, которые определены для элемента;

2) устанавливает тип данных каждого атрибута;

3) задает востребованность для каждого атрибута. Если атрибут не востребован, в объявлении списка атрибутов указывается, что должен делать процессор, если атрибут опущен. (В объявлении должно, например, содержаться значение атрибута по умолчанию, которое будет использовать процессор.)

5.6.1. Форма записи объявления списка атрибутов

Объявление списка атрибутов имеет следующую общую форму:

```
<!ATTLIST Имя ОпрАтр>
```

Здесь Имя представляет собой имя элемента, ассоциированного с атрибутом или атрибутами. ОпрАтр – это одно или несколько определений атрибутов, каждое из которых определяет один атрибут.

Определение атрибута имеет следующую форму записи:

```
Имя ОпрАтр ОбъявУмолч
```

Здесь Имя – это имя атрибута. ОпрАтр представляет собой тип атрибута, т. е. виды значений, которые могут быть присвоены атрибуту. (О типах атрибутов пойдет речь в следующем пункте.) ОбъявУмолч – это объявление по умолчанию, которое указывает на востребованность атрибута и содержит другую информацию.

Допустим, вы объявили тип элемента с именем FILM следующим образом:

```
<!ELEMENT FILM (TITLE, (STAR | NARRATOR |  
INSTRUCTOR) )>
```


На рис. 5.5 показано, как это будет отображено в браузере.

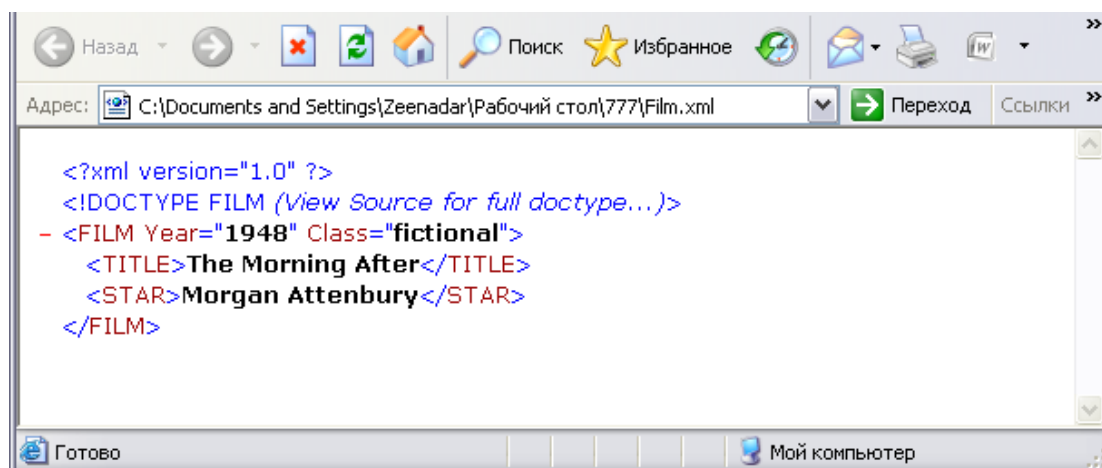


Рис. 5.5. Отображение в Internet Explorer 5 использования списка атрибутов

Для элемента FILM атрибуту Year присвоено значение «1948». Атрибут Class опущен; однако, поскольку этот атрибут имеет значение по умолчанию («fictional»), оно присваивается атрибуту, как если бы вы записали его в качестве значения атрибута.

⇒ *Примечание.* Если вы использовали для данного типа элемента более одного объявления списка атрибутов, то содержания двух объявлений объединяются. Если атрибут с заданным именем объявлен для одного и того же элемента несколько раз, первое объявление используется, а последующие – игнорируются.

5.6.2. Тип атрибута

Тип атрибута является вторым необходимым компонентом в определении атрибута. Он задает вид значений, которые вы можете присваивать атрибуту внутри документа (рис. 5.6).

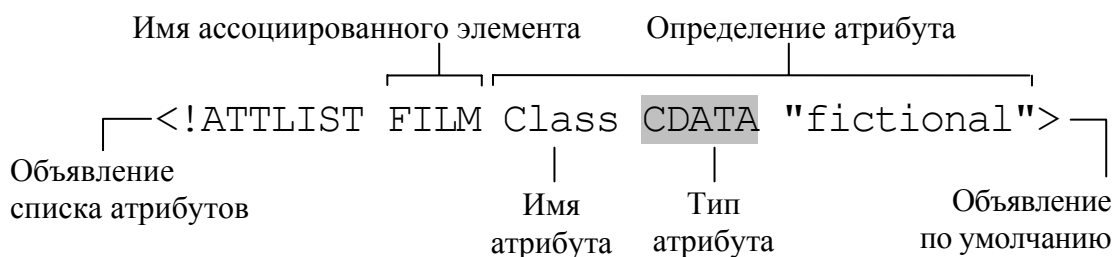


Рис. 5.6. Задание типа атрибута

Вы можете задавать тип атрибута тремя различными способами:

- строковый тип. Строковый тип атрибута может быть назначен любой строке в кавычках (литералу), которая отвечает общим правилам, описанным в 3.3.2 «Правила для корректного задания значений атрибутов». Вы должны объявить строковый тип атрибута с использованием ключевого слова CDATA, как это сделано для определения атрибута Class в следующем примере:

```
<!ATTLIST FILM Class CDATA "fictional">
```

- маркерный тип. Значения, которые вы можете присваивать атрибуту маркерного типа, имеют ряд ограничений;
- нумерованный тип. Для нумерованного типа атрибута вы можете присваивать одно значение или список определенных значений. Об этом типе вы узнаете в 5.8 «Задание нумерованных типов».

5.7. ЗАДАНИЕ МАРКЕРНОГО ТИПА

Как и любое значение атрибута, значение, которое вы присваиваете маркерному типу атрибута, должно представлять собой строку в кавычках, отвечающую общим правилам, описанным в 3.3.2 «Правила для корректного задания значений атрибутов».

Кроме того, значение должно отвечать определенному ограничению, которое вы задаете в описании атрибута с помощью соответствующего ключевого слова. Например, в приведенном ниже XML-документе для атрибута StockCode определен маркерный тип с использованием ключевого слова ID. (ID – это только одно из ключевых слов, которые вы можете использовать для объявления маркерного типа.) Это ключевое слово означает, что для каждого элемента атрибуту должно быть присвоено уникальное значение. (Например, присвоение товарного кода «S021» двум элементам ITEM не допускается.)

Листинг 5.4. Маркерный тип атрибутов

```
<?xml version="1.0"?>
<!DOCTYPE INVENTORY
[
  <!ELEMENT INVENTORY (ITEM*)>
  <!ELEMENT ITEM (#PCDATA)>
  <!ATTLIST ITEM StockCode ID #REQUIRED>
]>
```

```

<INVENTORY>
  <ITEM StockCode="S021">Peach Tea Pot</ITEM>
  <ITEM          StockCode="S034">Electric          Cofee
Grinder</ITEM>
  <ITEM StockCode="S086">Candy Thermometer</ITEM>
</INVENTORY>

```

На рис. 5.7 показано, как будет данный пример отображен в браузере.

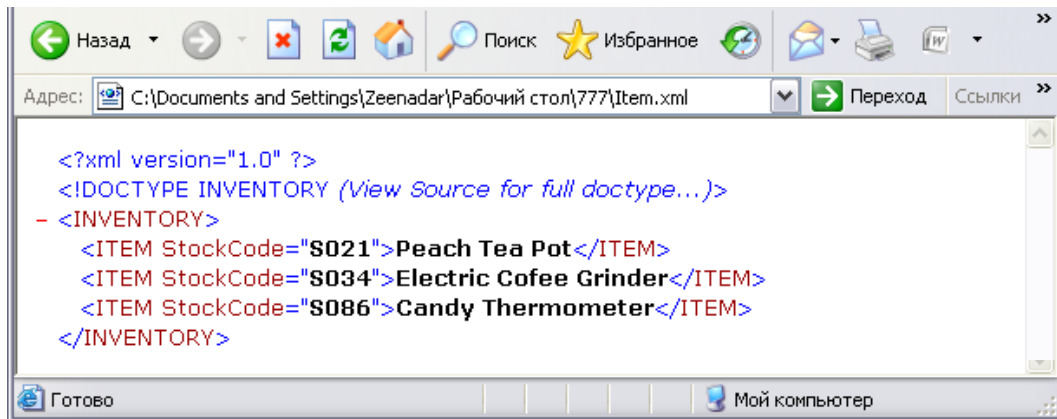


Рис. 5.7. Отображение в Internet Explorer 5 маркерного типа атрибутов

Ниже приведен полный список ключевых слов, которые вы можете использовать в определении маркерных типов атрибутов, и ограничения, которые они накладывают на значения атрибутов:

1) ID. Для каждого элемента атрибут должен иметь уникальное значение. Значение должно начинаться с буквы или символа подчеркивания (), за которыми могут идти или не идти другие буквы, цифры, точки (.), тире (–) или символы подчеркивания. Данный тип элемента может иметь только один атрибут типа ID, а в объявлении значения атрибута по умолчанию должно фигурировать #REQUIRED или #IMPLIED. Пример этого типа атрибута содержится в приведенном выше документе INVENTORY;

2) IDREF. Значение атрибута должно совпадать со значением атрибута элемента типа ID внутри документа. Другими словами, этот тип атрибута является ссылкой на уникальный идентификатор другого атрибута. Например, вы можете добавить атрибут IDREF с именем GoesWith к элементу ITEM:

```

<!ELEMENT ITEM (#PCDATA)>
<ATTLIST ITEM
  StockCode ID #REQUIRED  GoesWith IDREF #IMPLIED>

```

Далее вы можете использовать этот атрибут для ссылки на другой элемент ITEM:

```
<ITEM StockCode="S034">Electric Cofee Grinder</ITEM>
<ITEM StockCode="S047" GoesWith="S034">
Cofee Grinder Brush
</ITEM>
```

3) IDREFS. Данный тип атрибута похож на тип IDREF, но при этом значение может включать ссылки на несколько идентификаторов – разделенных пробелами – внутри строки в кавычках. Например, если вы назначите атрибуту GoesWith тип IDREFS таким образом:

```
<!ATTLIST ITEM      StockCode ID #REQUIRED      GoesWith
IDREFS #IMPLIED>
```

то можете использовать его в качестве ссылки на несколько других элементов:

```
<ITEM StockCode="S034">Electric Cofee Grinder</ITEM>
<ITEM StockCode="S039">
1 pound Breakfast Blend Cofee Beans
</ITEM>
<ITEM StockCode="S047" GoesWith="S034 S039">
Cofee Grinder Brush
</ITEM>
```

4) ENTITY. Значение атрибута должно совпадать с именем примитива, объявленного в DTD. Этот примитив не обрабатывается синтаксическим анализатором и ссылается на внешний файл, обычно содержащий не XML-данные.

Например, в DTD вы объявляете элемент с именем IMAGE, представляющий графическое изображение, и атрибут типа ENTITY с именем Source, указывающий на источник графических данных:

```
<!ELEMENT IMAGE EMPTY>
<ATTLIST IMAGE      Source ENTITY #REQUIRED>
```

Если вы объявили неанализируемый примитив с именем Logo, который содержит графические данные для изображения, вы можете присвоить этот примитив атрибуту Source элемента IMAGE в документе следующим образом:

```
<IMAGE Source="Logo"/>
```

5) ENTITIES. Этот тип атрибута похож на тип ENTITY, за исключением того, что значение может содержать имена нескольких неанализируемых примитивов – разделенных пробелами – внутри строки в кавычках. Например, если вы назначили атрибуту Source тип ENTITIES следующим образом:

```
<!ELEMENT IMAGE EMPTY>
<!ATTLIST IMAGE Source ENTITIES #REQUIRED>
```

то сможете использовать его для ссылки на несколько неанализируемых примитивов (допустим, примитивов, содержащих графические данные в альтернативных форматах), например, так:

```
<IMAGE Source="LogoGif LogoBmp"/>
```

(Здесь подразумевается, что LogoGif и LogoBmp – имена неанализируемых примитивов, которые были объявлены в DTD с помощью приемов, с которыми вы познакомитесь в главе 6.)

6) NMTOKEN. Это значение есть элементарное имя (name token), представляющее собой имя, которое состоит из одной или более букв, цифр, точек (.), тире (–) или символов подчеркивания (_). Элементарное имя может также содержать двоеточие (:), но не на первом месте. Например, если вы назначите атрибуту ISBN тип NMTOKEN следующим образом:

```
<!ELEMENT BOOK (#PCDATA)>
<!ATTLIST BOOK ISBN NMTOKEN #REQUIRED>
```

то сможете присвоить ему значение, начинающееся с цифры (цифры в качестве первых символов допустимы для типов NMTOKEN и NMTOKENS, но не для любых других маркерных типов):

```
<BOOK ISBN="9-99999-999-9">The Portrait of a Lady</BOOK>
```

7) NMTOKENS. Этот тип атрибута похож на тип NMTOKEN, но значение может содержать несколько элементарных имен – разделенных пробелами – внутри строки в кавычках. Например, если вы назначите атрибуту Codes тип NMTOKENS следующим образом:

```
<!ELEMENT SHIRT (#PCDATA)>
<!ATTLIST SHIRT Codes NMTOKENS #REQUIRED>
```

то сможете присвоить ему несколько значений в виде элементарных имен:

```
<SHIRT Codes="38 21 97">long sleeve Henley</SHIRT>
```

5.8. ЗАДАНИЕ НУМЕРОВАННЫХ ТИПОВ

Как любое значение атрибута, значение, которое вы присваиваете номерованному типу, должно представлять собой строку в кавычках, отвечающую правилам, описанным в 3.3.2 «Правила для корректного задания значений атрибутов». Помимо этого, значение должно совпадать с одним из имен, приведенных в списке типов атрибутов. Эти имена могут иметь одну из следующих двух форм записи:

– открывающая скобка, вслед за которой идет список элементарных имен, разделенных символами |, после чего следует закрывающая скобка. Напомним, что элементарное имя – это имя, которое состоит из одной или нескольких букв, цифр, точек (.), тире (–) или символов подчеркивания (_), а также может включать одно двоеточие (:), но не на первом месте. Например, если вы хотите ограничить значения атрибута Class словами «fictional», «instructional» или «documentary», то можете определить этот атрибут как номерованный тип следующим образом:

```
<!ATTLIST FILM
    Class (fictional | instructional | documentary)
    "fictional"
```

Вот законченный XML-документ, демонстрирующий использование атрибута Class.

Листинг 5.5. Атрибут Class

```
<?xml version="1.0"?>
<!DOCTYPE FILM
[
    <!ELEMENT FILM (TITLE, (STAR | NARRATOR | INSTRUCTOR) )>
    <!ATTLIST FILM
        Class (fictional | instructional | documentary)
            "fictional">

    <!ELEMENT TITLE (#PCDATA)>
    <!ELEMENT STAR (#PCDATA)>
    <!ELEMENT NARRATOR (#PCDATA)>
    <!ELEMENT INSTRUCTOR (#PCDATA)>
]>
<FILM Class="instructional">
    <TITLE>The Use and Care of XML</TITLE>
    <NARRATOR>Michael Young</NARRATOR>
</FILM>
```

На рис. 5.8 показано, как это будет отображено в браузере.

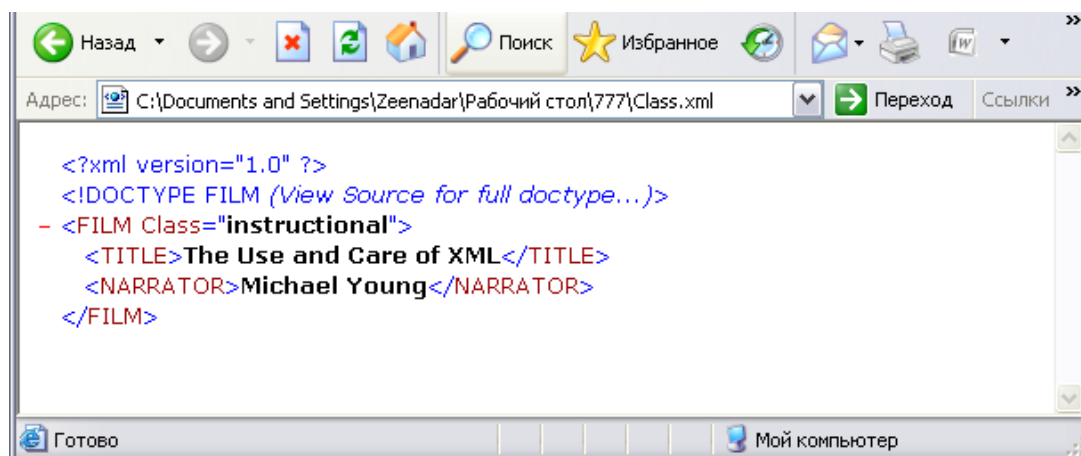


Рис. 5.8. Отображение в Internet Explorer 5 атрибута Class

Если вы опустили атрибут Class, ему будет по умолчанию присвоено значение «fictional». Присвоение атрибуту Class значения, отличного от «fictional», «instructional» или «documentary», приведет к ошибке;

– ключевое слово NOTATION, за которым идет пробел, затем открывающая скобка, список имен нотаций, разделяемых символами |, после чего следует закрывающая скобка. Каждое из этих имен должно точно соответствовать имени нотации, объявленному в DTD. Нотация описывает формат данных или идентифицирует программу, применяемую для обработки определенного формата.

Например, в вашем DTD объявлены нотации HTML, SGML и RTF. Тогда вы можете ограничить значения атрибута Format одним из этих имен нотаций с помощью следующего объявления:

```
<!ELEMENT EXAMPLE_DOCUMENT (#PCDATA)>
<!ATTLIST EXAMPLE_DOCUMENT
    Format NOTATION (HTML | SGML | RTF) #REQUIRED>
```

В дальнейшем вы можете использовать элемент Format для указания формата определенного элемента EXAMPLE_DOCUMENT, как в следующем примере.

Листинг 5.6. Элемент Format

```
<EXAMPLE_DOCUMENT Format="HTML">
    <![CDATA[
        <HTML>
        <HEAD>
```



```

<TITLE>Mike's Home Page</TITLE>
</HEAD>
<BODY>
<P>Welcome!</P>
</BODY>
</HTML>
]]>
</EXAMPLE_DOCUMENT>

```

Присвоение атрибуту Format значения, отличного от «HTML», «SGML» или «RTF», приведет к ошибке.

5.9. ОБЪЯВЛЕНИЕ ЗНАЧЕНИЯ ПО УМОЛЧАНИЮ

Объявление значения по умолчанию – это третий и последний обязательный компонент в определении атрибута. Оно задает, является ли атрибут обязательным, и если нет, указывает, что должен предпринимать процессор в случае, когда атрибут опущен. Так, объявление должно обеспечить значение атрибута по умолчанию, которое будет использовать процессор в том случае, если атрибут отсутствует (рис. 5.9).

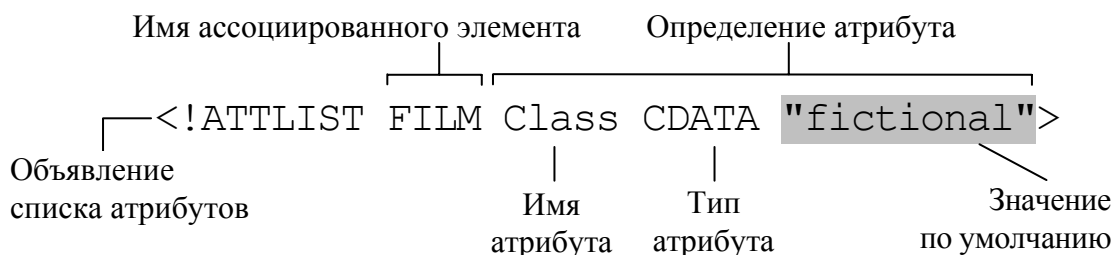


Рис. 5.9. Объявление значения по умолчанию

Объявление значения по умолчанию может иметь следующие четыре формы:

1) #REQUIRED. При этой форме вы должны задать значение атрибута для каждого элемента ассоциированного типа. Например, следующее объявление указывает, что вы должны присвоить значение атрибуту Class внутри начального тега каждого элемента FILM в документе:

```
<!ATTLIST FILM Class CDATA #REQUIRED>
```

2) #IMPLIED. Эта форма указывает, что вы можете либо включить, либо опустить атрибут для элемента ассоциированного типа, а также, что если вы опускаете атрибут, то никакое значение по умолчанию процессору не передается. Например, следующее объявление указывает, что присвоение значения атрибуту Class внутри элемента FILM является необязательным и что в документе не представлено значение Class по умолчанию:

```
<!ATTLIST FILM Class CDATA #IMPLIED>
```

3) AttValue, где AttValue – значение атрибута по умолчанию. При такой форме вы можете либо включить, либо опустить атрибут для элемента ассоциированного типа. Если вы опускаете атрибут, процессор использует значение по умолчанию, как если бы вы включили атрибут и задали это значение.

Задаваемое значение по умолчанию, конечно, должно соответствовать заданному типу атрибута. Например, следующее объявление присваивает значение по умолчанию «fictional» атрибуту Class:

```
<!ATTLIST FILM Class CDATA "fictional">
```

Согласно этому объявлению, следующие два элемента эквивалентны:

```
<FILM>The Graduate</FILM>  
<FILM Class="fictional">The Graduate</FILM>
```

4) #FIXED AttValue, где AttValue – значение атрибута по умолчанию. При такой форме вы можете либо включить, либо опустить атрибут для элемента ассоциированного типа. Если вы опускаете атрибут, процессор будет использовать значение, заданное по умолчанию; если вы включаете атрибут, вы должны задать значение по умолчанию. Например, следующее объявление присваивает фиксированное значение по умолчанию атрибуту Class:

```
<!ATTLIST FILM Class CDATA #FIXED "documentary">
```

В соответствии с этим объявлением следующие два эквивалентных элемента будут корректными:

```
<FILM>The Making of XML</FILM>  
<FILM Class="documentary">The Making of XML</FILM>
```


в то время как следующий элемент будет некорректным:

```
<!-- Некорректный элемент! -->  
<FILM Class="instructional">The Making of XML</FILM>
```

5.10. ИСПОЛЬЗОВАНИЕ ВНЕШНИХ ПОДМНОЖЕСТВ DTD

Описания типа документа, рассмотренные нами в этой главе, полностью содержатся внутри объявления типа документа в составе документа. Такой тип DTD называется внутренним подмножеством DTD.

Вы также можете поместить все или часть DTD-документа в отдельный файл, а затем сослаться на этот файл из объявления типа документа. DTD, или часть DTD, содержащаяся в отдельном файле, называется внешним подмножеством DTD.

 *Примечание.* Применение внешнего подмножества DTD имеет смысл главным образом для DTD, которые являются общими для целой группы документов. Каждый документ может ссылаться на один файл DTD (или копию этого файла) как на внешнее подмножество DTD. При этом вам не надо копировать содержимое DTD в каждый использующий его документ, а также облегчается внесение изменений в DTD. Как вам известно из главы 1, многие стандартные XML-приложения основаны на общем DTD, включаемом во все XML-документы, которые отвечают этому приложению.

5.10.1. Использование только внешнего подмножества DTD

Чтобы использовать только внешнее подмножество DTD, опустите блок объявлений разметки, ограниченных квадратными скобками ([]), и вместо этого включите ключевое слово SYSTEM, после которого в кавычках должно следовать описание местонахождения отдельного файла, содержащего DTD. Рассмотрим, например, документ SIMPLE, используемый ранее в этой главе и имеющий внутреннее подмножество DTD.

Листинг 5.7. Документ SIMPLE

```
<?xml version="1.0"?>
<!DOCTYPE SIMPLE
  [ <!ELEMENT SIMPLE ANY>
  ]>
<SIMPLE> This is an extremely simplistic XML
document. </SIMPLE>
```

Если в этом документе используется внешнее подмножество DTD, он будет иметь следующий вид.

Листинг 5.8. Документ SIMPLE с внешним подмножеством DTD

```
<?xml version="1.0"?>
<!DOCTYPE SIMPLE SYSTEM "Simple.dtd">
<SIMPLE> This is an extremely simplistic XML
document. </SIMPLE>
```

Файл Simple.dtd должен иметь следующее содержимое:

```
<!ELEMENT SIMPLE ANY>
```

Файл, содержащий внешнее подмножество DTD, может включать любые объявления разметки, которые могут быть включены во внутреннее подмножество DTD.

Описание местонахождения файла (в данном примере Simple.dtd) называется системным литералом. Он может быть заключен в одинарные (') или двойные (") кавычки и содержать любые символы, за исключением символов кавычек, используемых как ограничители.

Системный литерал задает унифицированный идентификатор ресурса (Uniform Resource Identifier – URI) файла, содержащего внешнее подмножество DTD. В настоящее время URI практически аналогичен стандартному Internet-адресу, известному как унифицированный указатель ресурса (Uniform Resource Locator – URL). Вы можете использовать полностью прописанный URI, подобно следующему:

```
<!DOCTYPE SIMPLE SYSTEM "HYPERLINK "http://bogus.com/
dtds/Simple.dtd"
http://bogus.com/dtds/Simple.dtd">
```

Или вы можете применять частичный URI, который задает местонахождение относительно местонахождения XML-документа, содержащего URI, например:

```
<!DOCTYPE SIMPLE SYSTEM "Simple.dtd">
```

5.10.2. Использование и внешних, и внутренних подмножеств DTD

Чтобы использовать и внешнее, и внутреннее подмножество DTD, следует применять ключевое слово SYSTEM вместе с системным литералом, задающим местонахождение файла с внешним подмножеством DTD, после чего внутри квадратных скобок ([]) следует объявление разметки внутреннего подмножества DTD.

Вот пример простого XML-документа, имеющего как внутреннее, так и внешнее подмножество DTD.

Листинг 5.9. Внешнее и внутреннее подмножества DTD

```
<?xml version="1.0"?>
<!DOCTYPE BOOK SYSTEM "Book.dtd"
  [<!ATTLIST BOOK ISBN CDATA #IMPLIED Year CDATA
"2000">
  <!ELEMENT TITLE (#PCDATA)>
  ]>
<BOOK Year="1998">
  <TITLE>The Scarlet Letter</TITLE>
</BOOK>
```

Вот содержимое файла Book.dtd, в котором хранится внешнее подмножество DTD:

```
<!ELEMENT BOOK ANY>
<!ATTLIST BOOK ISBN NMTOKEN #REQUIRED>
```

Если вы используете внешнее и внутреннее подмножества DTD, XML-процессор объединяет их содержимое следующим образом:

- в общем случае он осуществляет слияние двух подмножеств, чтобы сформировать полный DTD. В рассмотренном примере объединенный DTD определяет два элемента (TITLE и BOOK), а также два атрибута для элемента BOOK (ISBN и Year);

- однако в случае, если атрибут с одним и тем же именем и типом элемента объявляется более одного раза, XML-процессор использует первое объявление и игнорирует все последующие;

- внутреннее подмножество DTD имеет приоритет перед внешним подмножеством DTD (даже если ссылка на внешнее подмножество идет первой в объявлении типа документа). Таким образом, любой атрибут (или примитив), определенный во внутреннем подмножестве, доминирует над атрибутом с тем же именем и типом элемента, объявленным во внешнем подмножестве. В примере XML-процессор считает, что атрибут ISBN имеет тип CDATA и объявление значения по умолчанию #IMPLIED, поэтому следующий элемент (в котором не указан ISBN) является корректным:

```
<BOOK Year="1850">
  <TITLE>The Scarlet Letter</TITLE>
</BOOK>
```

Способ объединения внутреннего и внешнего подмножеств DTD XML-процессором дает вам возможность использовать общий DTD в качестве внешнего подмножества DTD, а затем адаптировать (или субклассировать, как говорят программисты) DTD

для конкретного документа путем включения внутреннего подмножества. Ваше внутреннее подмножество может добавлять элементы, атрибуты или примитивы, т. е. также может изменять определения атрибутов или примитивов.

5.10.3. Условия игнорирования разделов внешнего подмножества DTD

Вы можете заставить XML-процессор игнорировать часть внешнего подмножества DTD с помощью раздела IGNORE. Вы можете, например, использовать раздел IGNORE при разработке документа с целью временного отключения альтернативного или необязательного блока объявлений разметки. При этом вам не нужно удалять строки, а затем повторно их вставлять. Раздел IGNORE начинается с символов `<![IGNORE[` и заканчивается символами `]]>`.

На рис. 5.10 представлен пример полного описания внешнего подмножества DTD, включающего раздел IGNORE.

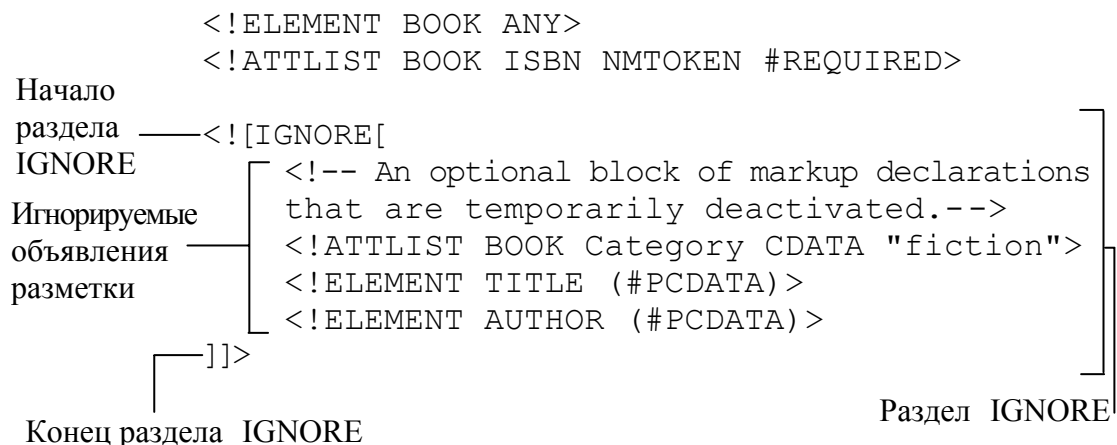


Рис. 5.10. Полное описание внешнего подмножества DTD

Если вы хотите временно восстановить блок объявлений разметки в разделе IGNORE, вам достаточно просто заменить ключевое слово IGNORE на INCLUDE, не удаляя при этом символы-ограничители (`<![`, `[` и `]]>`).

Листинг 5.10. Замена IGNORE на INCLUDE

```

<![ INCLUDE [
    <!-- Необязательный блок объявлений разметки,
         который временно восстановлен. -->
    <![ATTLIST BOOK Category CDATA "fiction">
  
```

```

<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT AUTHOR (#PCDATA)
]>

```

Впоследствии вы можете снова быстро отключить раздел, вернув заголовок IGNORE. Раздел INCLUDE, вложенный в раздел IGNORE, также игнорируется.

⇒ *Примечание.* Вы можете использовать разделы IGNORE и INCLUDE только во внешнем подмножестве DTD либо во внешнем параметрическом примитиве.

Далее вы попытаетесь на практике применить полученные при изучении этой главы знания, преобразовав корректно сформированный документ в валидный.

? ЗАДАНИЯ

1. Сделайте документ валидным

В вашем текстовом редакторе откройте документ Inventory.xml, созданный вами в главе 2.

Непосредственно перед элементом Документ – с именем INVENTORY – введите следующее объявление типа документа:

```

<![INCLUDE [
  <!-- Необязательный блок объявлений разметки,
        который временно восстановлен. -->
  <!ATTLIST BOOK  Category CDATA "fiction">
  <!ELEMENT TITLE (#PCDATA)>
  <!ELEMENT AUTHOR (#PCDATA)
]]>

```

Обратите внимание, что имя следующего за DOCTYPE ключевого слова совпадает с именем элемента Документ – INVENTORY. DTD состоит только из внутреннего подмножества, которое определяет элементы и атрибуты документа следующим образом:

- элемент Документ – INVENTORY имеет содержимое и может включать ни одного или несколько дочерних элементов BOOK;
- элемент BOOK также имеет содержимое, которое должно включать строго по одному из следующих элементов в порядке, перечисленном в объявлении документа: TITLE, AUTHOR, BINDING, PAGES и PRICE;

– элемент TITLE имеет смешанное содержимое и может включать символьные данные вместе с ни одним или с несколькими элементами SUBTITLE;

– элементы AUTHOR, BINDING, PAGES и PRICE также имеют смешанное содержимое. Эти элементы, однако, могут включать только символьные данные без дочерних элементов;

– элемент BOOK имеет атрибут нумерованного типа с именем InStock, который является обязательным атрибутом и может принимать значения либо «yes», либо «no»;

– элемент AUTHOR имеет атрибут строкового типа с именем Born, который является необязательным и не имеет значения по умолчанию.

2. Преобразуйте валидный документ

Добавьте следующий дочерний элемент SUBTITLE в элемент TITLE для книги Моби-Дик:

```
<BOOK>
  <TITLE>Moby-Dick</TITLE>
  <SUBTITLE>Or, the Whale</SUBTITLE>
```

Добавьте обязательный атрибут InStock каждому элементу BOOK, присвоив ему значение «yes» или «no», как показано ниже:

```
<BOOK InStock="yes">
  <TITLE>The Adventures of Huckleberry Finn</TITLE>
  <AUTHOR>Mark Twain</AUTHOR>
  <BINDING>mass market paperback</BINDING>
  <PAGES>298</PAGES>
  <PRICE>$5.49</PRICE>
</BOOK>
```

Добавьте необязательный элемент Born к одному или нескольким элементам. Хотя вы можете присвоить этому атрибуту любую синтаксически правильную строку в кавычках, в данном случае он должен хранить дату рождения автора. Пример:

```
<AUTHOR Born="1835">Mark Twain</AUTHOR>
```

Чтобы отразить новое имя файла, которое вы собираетесь присвоить, измените комментарий в начале документа

```
<!-- Имя файла: Inventory.xml -->
```

на

```
<!-- Имя файла: Inventory Valid.xml -->
```


Выберите команду Save As (Сохранить как) вашего текстового редактора, чтобы сохранить копию модифицированного документа под именем Inventory Valid.xml.

Если вы хотите проверить валидность вашего документа, воспользуйтесь сценарием проверки XML-документа на валидность, приведенным в пункте «Проверка валидности XML-документа» (см. на с. 291) в главе 9.

На рис. 5.11 показано, как будет отображен в браузере следующий код программы.

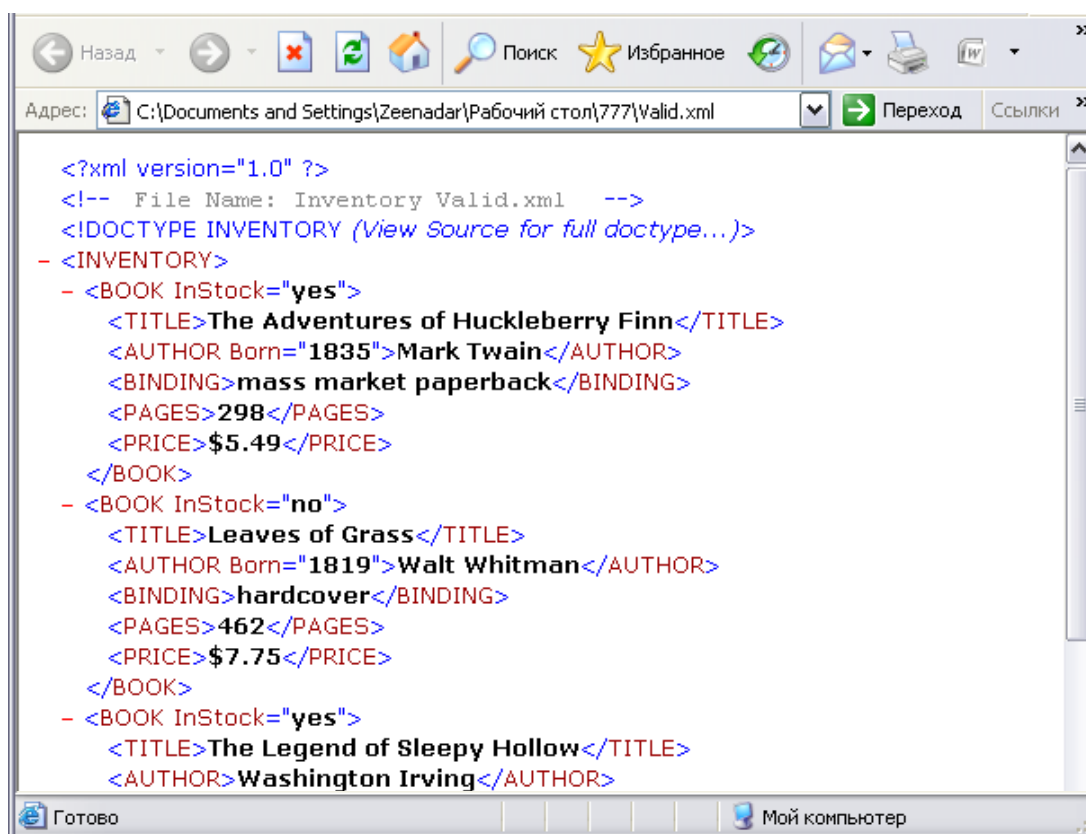


Рис. 5.11. Отображение в Internet Explorer 5 проверки валидности XML-документа

Листинг 5.11. Проверка валидности XML-документа

```
<?xml version="1.0"?>
<!-- File Name: Inventory Valid.xml -->
<!DOCTYPE INVENTORY
[
  <!ELEMENT INVENTORY (BOOK)*>
  <!ELEMENT BOOK (TITLE, AUTHOR, BINDING, PAGES, PRICE)>
  <!ATTLIST BOOK    InStock (yes|no) #REQUIRED>
```

```

<!ELEMENT TITLE (#PCDATA | SUBTITLE)*>
<!ELEMENT SUBTITLE (#PCDATA)>
<!ELEMENT AUTHOR (#PCDATA)>
<!ATTLIST AUTHOR    Born CDATA #IMPLIED>
<!ELEMENT BINDING (#PCDATA)>
<!ELEMENT PAGES (#PCDATA)>
<!ELEMENT PRICE (#PCDATA)>
]>
<INVENTORY>
  <BOOK InStock="yes">
    <TITLE>The Adventures of Huckleberry Finn</TITLE>
    <AUTHOR Born="1835">Mark Twain</AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>298</PAGES>
    <PRICE>$5.49</PRICE>
  </BOOK>
  <BOOK InStock="no">
    <TITLE>Leaves of Grass</TITLE>
    <AUTHOR Born="1819">Walt Whitman</AUTHOR>
    <BINDING>hardcover</BINDING>
    <PAGES>462</PAGES>
    <PRICE>$7.75</PRICE>
  </BOOK>
  <BOOK InStock="yes">
    <TITLE>The Legend of Sleepy Hollow</TITLE>
    <AUTHOR>Washington Irving</AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>98</PAGES>
    <PRICE>$2.95</PRICE>
  </BOOK>
  <BOOK InStock="yes">
    <TITLE>The Marble Faun</TITLE>
    <AUTHOR Born="180">Nathaniel Hawthorne</AUTHOR>
    <BINDING>trade paperback</BINDING>
    <PAGES>473</PAGES>
    <PRICE>$10.95</PRICE>
  </BOOK>
  <BOOK InStock="no">
    <TITLE>Moby-Dick
    <SUBTITLE>Or, the Whale</SUBTITLE>
    </TITLE>
    <AUTHOR Born="1819">Herman Melville</AUTHOR>
    <BINDING>hardcover</BINDING>

```

```

        <PAGES>724</PAGES>
        <PRICE>$9.95</PRICE>
    </BOOK>
    <BOOK InStock="yes">
        <TITLE>The Portrait of a Lady</TITLE>
        <AUTHOR>Henry James</AUTHOR>
        <BINDING>mass market paperback</BINDING>
        <PAGES>256</PAGES>
        <PRICE>$4.95</PRICE>
    </BOOK>
    <BOOK InStock="yes">
        <TITLE>The Scarlet Letter</TITLE>
        <AUTHOR>Nathaniel Hawthorne</AUTHOR>
        <BINDING>trade paperback</BINDING>
        <PAGES>253</PAGES>
        <PRICE>$4.25</PRICE>
    </BOOK>
    <BOOK InStock="no">
        <TITLE>The Turn of the Screw</TITLE>
        <AUTHOR>Henry James</AUTHOR>
        <BINDING>trade paperback</BINDING>
        <PAGES>384</PAGES>
        <PRICE>$3.35</PRICE>
    </BOOK>
</INVENTORY>
```

Глава 6. ПРИМИТИВЫ

6.1. ОПРЕДЕЛЕНИЕ И ИСПОЛЬЗОВАНИЕ ПРИМИТИВОВ

В этой главе вы прежде всего познакомитесь с основной терминологией, используемой для примитивов, а также с различными способами классификации примитивов. Затем вы узнаете, как объявлять каждый из примитивов различных типов и как вставлять примитивы в ваш документ в необходимом месте. Далее вы получите представление о том, как использовать возможности XML, которые позволяют вам вставлять любые типы символов в любом контексте: ссылки на символ и примитивы. В конце главы имеется упражнение, которое позволит вам попрактиковаться в использовании примитивов в полном XML-документе.

6.1.1. Определения и классификация примитивов

Механизм примитивов в XML является средством повышения производительности, а также способом встраивать различные типы данных в ваш XML-документ. В XML-документе вы можете определить часто используемый блок XML-текста как примитив, что позволит вам быстро вставлять текст в нужное место. Вы можете также определить внешний файл как примитив, чтобы иметь возможность включать данные файла в вашем документе; эти данные могут содержать XML-текст, другой текст либо нетекстовые данные. Вы определяете примитив в описании типа документа (DTD) с использованием синтаксиса, аналогичного тому, который применяется для объявления элемента или атрибута в валидном XML-документе. О DTD и объявлениях типа документа, содержащего их, говорилось в главе 5.

6.1.2. Внешнее подмножество DTD

В спецификации XML термин «примитив» (entity) в широком смысле относится к любому из следующих типов единиц хранения информации для XML-документов:

1) собственно XML-документ как целое;
2) внешний файл, определенный как внешний примитив в DTD и допускающий использование посредством ссылки;

3) строка в кавычках, определенная как внутренний примитив в DTD и допускающая использование с помощью ссылки.

Заметим, что первые два типа единиц хранения информации являются файлами, а последний – строкой символов, заключенных в кавычки.

В этой главе термин «примитив» используется в узком смысле, а именно, для обозначения внешнего файла или строки в кавычках, определенных как примитив в DTD документа и допускающих использование в документе посредством ссылок на примитивы. Например, следующее DTD определяет внешний файл `Topics.xml` (этот файл содержит список тем в статье, которая включена в документ) как внешний примитив с именем `topics`, а также строку в кавычках («A Short History of XML») как внешний примитив с именем `title`.

Листинг 6.1. Внешний примитив

```
<!DOCTYPE ARTICLE
[
  <!ELEMENT ARTICLE (TITLEPAGE, INTRODUCTION, SECTION*)>
  <!ELEMENT TITLEPAGE (#PCDATA)>
  <!ELEMENT INTRODUCTION (#PCDATA)>
  <!ELEMENT SECTION (#PCDATA)>
  <!ENTITY topics SYSTEM "Topics.xml">
  <!ENTITY title "A short History of XML">
]
<
```

Впоследствии вы можете вставить полный список тем в любое нужное вам место статьи (например, в аннотацию, введение или заключение), просто включив ссылку на примитив `&topics`.

Листинг 6.2. Ссылка на примитив &topics

```
<INTRODUCTION>
  This article will cover the following topics:
  &topics;
</INTRODUCTION>
```

Вы можете вставить название статьи в любое место, включив ссылку на примитив `&title`.

Листинг 6.3. Ссылка на примитив &title

```
<TITLEPAGE>
  Title:  &title;
  Author: Michael Young
</TITLEPAGE>
```

Механизм примитивов наиболее полезен при наличии часто используемых фрагментов XML-текста. Например, если название статьи многократно фигурирует по ее тексту, использование примитива (как в предыдущем примере) позволит сократить время набора, добиться однородности и облегчить внесение изменений в название. Вы можете изменить текст названия, встречающегося в различных листах статьи, просто отредактировав объявление примитива в DTD. Например:

```
<!ENTITY title "A Long History of XML">
  <! -- Модифицированное объявление примитива. -->
```

Если вы знакомы с программированием, то легко уловите сходство между механизмом использования примитивов XML и определением констант в языках программирования (например, объявления с помощью инструкции `#define` в C++).

Механизм примитивов также необходим при включении неXML-данных в XML-документ (например, графические данные для изображения).

6.1.3. Типы примитивов

Существует множество разновидностей примитивов. Материал, излагаемый в этом пункте, может показаться довольно абстрактным (прежде, чем вы познакомитесь с деталями и рассмотрите примеры), возвращение к этой информации впоследствии позволит значительно облегчить восприятие примитивов.

Примитивы классифицируются по трем признакам:

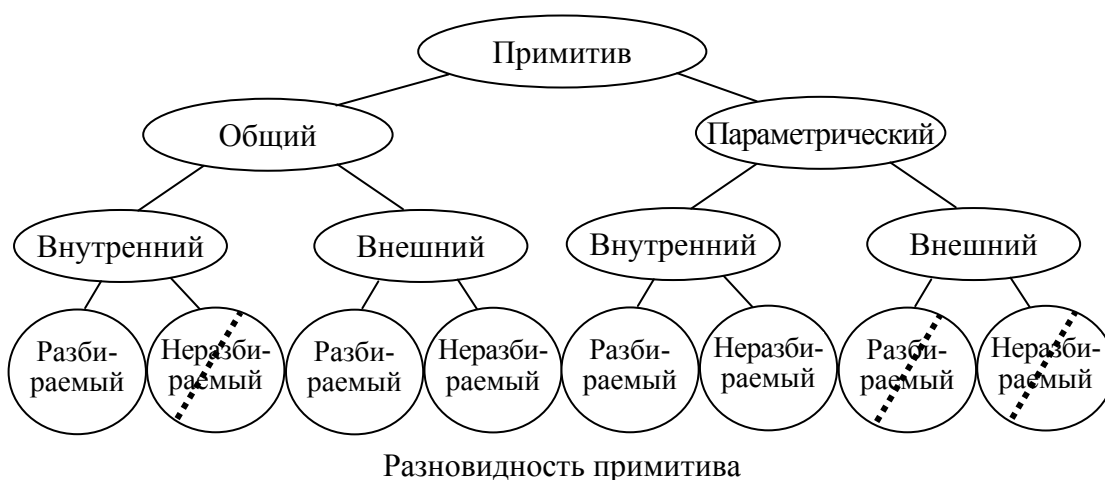
- общие и параметрические. Общий примитив включает содержимое документа, т. е. XML-текст либо другие текстовые или нетекстовые данные, которые вы можете использовать внутри элемента Документ. Оба примера примитивов, рассмотренных в предыдущем пункте (`title` и `topics`), относятся к общим примитивам. Параметрический примитив содержит XML-текст, который может быть помещен в DTD. В спецификации XML термин «примитив» относится к общим примитивам;

– внутренние и внешние. Внутренний примитив содержится внутри строки в кавычках (примитив `title` в предыдущем пункте). Внешний примитив содержится в отдельном файле (примитив `topics` в предыдущем пункте);

– разбираемые или неразбираемые. Разбираемый примитив содержит XML-текст (символьные данные, разметка или то и другое). Когда вы вставляете ссылку на разбираемый примитив в документ, ссылка замещается содержимым примитива (замещающий текст), который становится составной частью документа. Синтаксический анализатор XML разбирает (сканирует) содержимое примитива точно так же, как он сканирует непосредственно введенный в документ текст. Оба примера примитивов, рассмотренных в предыдущем пункте (`title` и `topics`), являются разбираемыми примитивами.

Неразбираемый примитив может содержать любой тип данных: XML-данные или, что чаще, неXML-данные. НеXML-данные могут представлять собой либо текстовые данные (например, название), либо нетекстовые данные (например, графические данные для изображения). Поскольку неразбираемый примитив обычно не содержит XML, его содержимое нельзя непосредственно вставить в документ путем ссылки на примитив. Тем не менее вы можете связать с именем примитива атрибут типа `ENTITY` или `ENTITIES`, чтобы приложение получило доступ к имени примитива и его описанию, а также могло работать с его данными.

Поскольку примитивы классифицируются по этим трем признакам и в каждой классификации имеются две категории, теоретически существует восемь потенциальных типов примитивов, как показано на рисунке.



Однако три типа примитивов из этих восьми в XML не поддерживаются (на рисунке они зачеркнуты). Следовательно, реально в XML имеется только пять типов примитивов:

- 1) общие внутренние разбираемые;
- 2) общие внешние разбираемые;
- 3) общие внешние неразбираемые;
- 4) параметрические внутренние разбираемые;
- 5) параметрические внешние разбираемые.

О том, как определять и использовать их, вы узнаете в этой главе.

6.2. ОБЪЯВЛЕНИЕ ОБЩИХ ПРИМИТИВОВ

Примитив создается путем объявления его в DTD документа. Вы объявляете примитив с использованием разновидности объявления разметки, схожей с той, которая применяется для объявления элементов и атрибутов. В последующих пунктах вы узнаете, как объявлять каждый из типов общих примитивов.

6.2.1. Объявление общего внутреннего разбираемого примитива

Объявление общего внутреннего разбираемого примитива имеет следующую форму записи:

```
<!ENTITY ИмяПримитива ЗначениеПримитива>
```

Здесь ИмяПримитива есть имя примитива. Вы можете выбрать любое имя, следуя следующим правилам:

- имя должно начинаться с буквы или с символа подчеркивания (`_`), после чего может следовать ни одна или несколько букв, цифр, точек (`.`), тире (`–`) или символов подчеркивания;
- примитив может иметь такое же имя, что и параметрический примитив в документе. (Общие и параметрические примитивы занимают различные пространства имен.) Примитив также может иметь такое же имя, как и элемент или атрибут;
- помните, что для всего текста внутри разметки имеет значение регистр, в котором набраны символы. Это относится и к именам примитивов. Так, примитив с именем `Bowser` и примитив с именем `bowser` будут считаться различными.

ЗначениеПримитива есть значение примитива. Значение, которое вы присваиваете общему примитиву, представляет собой группу символов, заключенных в кавычки, которая также носит название литерал. Вы можете присвоить любое значение типа литерал общему внутреннему примитиву, соблюдая при этом следующие правила:

1) строка должна быть заключена в одинарные (') или двойные (") кавычки;

2) строка не может содержать тот же символ кавычек, который используется в качестве ограничителей;

3) строка не может содержать символ амперсанда (&), если только он не применяется в качестве первого символа в указании ссылки на символ или на общий примитив. Строка также не должна содержать символ процентов (%);

4) содержимое строки должно быть корректным для места, в которое вы предполагаете вставить примитив. Например, если вы помещаете примитив внутрь элемента, он должен содержать один или несколько компонентов, которые могут быть корректно вставлены в другие элементы. Либо, если вы вставляете примитив внутрь значения атрибута, он должен содержать символы, которые являются допустимыми для значений атрибута. Далее в этой главе вы узнаете, куда можно помещать общие внутренние разбираемые примитивы.

Например, следующее DTD определяет общий внутренний разбираемый примитив с именем title.

Листинг 6.4. Внутренний разбираемый примитив title

```
<!DOCTYPE ARTICLE
[
  <!ELEMENT ARTICLE (TITLEPAGE, INTRODUCTION, SECTION*)>
  <!ELEMENT TITLEPAGE (#PCDATA | SUBTITLE)*>
  <!ELEMENT SUBTITLE (#PCDATA)>
  <!ELEMENT INTRODUCTION (#PCDATA)>
  <!ELEMENT SECTION (#PCDATA)>
  <!ENTITY title
    "The Story of XML
  <SUBTITLE>The Future Language of the Internet</SUBTITLE>">
]>
```

Примитив title содержит символьные данные плюс элемент SUBTITLE. В соответствии с объявлением в DTD это содержимое может быть корректно вставлено только в элемент TITLEPAGE, как показано ниже:

```
<TITLEPAGE>
  Title:  &title;
  Author: Michael Young
</TITLEPAGE>
```

XML-процессор заменит ссылку на примитив (&title;) содержимым примитива и обработает содержимое, как если бы вы непосредственно набрали его в документе в позиции ссылки, подобно следующему:

```
<TITLEPAGE>
  Title:  The Story of XML
  <SUBTITLE>The Future Language of the Internet</SUBTITLE>
  Author: Michael Young
</TITLEPAGE>
```

6.2.2. Объявление общего внешнего разбираемого примитива

Объявление общего внешнего разбираемого примитива имеет следующую форму записи:

```
<!ENTITY ИмяПримитива SYSTEM СистемЛитерал>
```

Здесь ИмяПримитива есть имя примитива. Вы можете выбрать любое имя, следуя правилам задания имен для общих примитивов, приведенным в предыдущем пункте.

СистемЛитерал – системный литерал, который описывает местонахождение файла, содержащего данные примитива. Системный литерал может быть ограничен одинарными (') или двойными (") кавычками и содержать любые символы, кроме символа кавычек, используемого в качестве ограничителя.

Системный литерал задает унифицированный идентификатор ресурса (URI) файла, содержащего данные примитива. На сегодняшний день URI практически то же самое, что стандартный Internet-адрес, известный как унифицированный указатель ресурса (URL). Вы можете использовать либо полный URI, например:

```
<!ENTITY abstract SYSTEM "HYPERLINK
'http://bogus.com/documents/Abstract.xml '
http://bogus.com/documents/Abstract.xml">
```

либо частичный URI, который задает местонахождение относительно местонахождения XML-документа, содержащего URI, например:

```
<!ENTITY abstract SYSTEM ":Abstract.xml">
```

Относительные URI в XML-документах работают аналогично относительным URL для HTML-страниц. Файл внешнего примитива может содержать только те составляющие, которые могут быть корректно вставлены в элемент. Вы можете поместить общий внешний разбираемый примитив только внутрь содержимого элемента.

Например, следующее DTD определяет внешний файл Topics.xml как общий внешний разбираемый примитив.

Листинг 6.5. Внешний разбираемый примитив

```
<!DOCTYPE ARTICLE
[
  <!ELEMENT ARTICLE (TITLEPAGE, INTRODUCTION, SECTION*)>
  <!ELEMENT TITLEPAGE (#PCDATA)>
  <!ELEMENT INTRODUCTION ANY>
  <!ELEMENT SECTION (#PCDATA)>
  <!ENTITY topics SYSTEM "Topics.xml">
]
>
```

Ниже приведено содержимое файла Topics.xml.

Листинг 6.6. Файл Topics.xml

```
<HEADING>Topics</HEADING>
The Need for XML
The Official Goals of XML
Standard XML Applications
Real-World Uses for XML
```

Этот типичный файл внешнего примитива содержит два пункта, которые вы можете включить в XML-элемент: вложенный элемент и блок символьных данных. Его содержимое может быть корректно вставлено в элемент INTRODUCTION (который имеет любой тип содержимого), как показано в следующем примере:

```
<INTRODUCTION>
  Here's what this article covers:
  &topics;
</INTRODUCTION>
```

6.2.3. Объявление общего внешнего неразбираемого примитива

Объявление общего внешнего неразбираемого примитива имеет следующую форму записи:

```
<!ENTITY ИмяПримитива SYSTEM СистемЛитерал NDATA ИмяНотации>
```

Здесь ИмяПримитива есть имя примитива. Вы можете выбрать любое имя.

СистемЛитерал – системный литерал, который описывает местонахождение файла, содержащего данные примитива. Он действует точно так же, как и системный литерал для описания местоположения общего внешнего разбираемого примитива.

⇒ *Примечание.* Ключевое слово NDATA указывает, что файл примитива содержит неразбираемые данные (они не обрабатываются синтаксическим анализатором).

ИмяНотации есть имя нотации, объявленной в DTD. Нотация описывает формат данных, содержащихся в файле примитива, или указывает на местонахождение программы, которая может обрабатывать эти данные. Об объявлении нотации будет говориться в следующей подглаве.

Файл неразбираемого внешнего примитива может содержать любой тип текста или нетекстовые данные. Они должны, конечно, соответствовать описанию формата, определяемого соответствующей нотацией.


Например, DTD в следующем XML-документе определяет файл Faun.gif (который содержит рисунок обложки книги) как общий внешний неразбираемый примитив с именем faun. Имя нотации этого примитива – GIF указывает на местонахождение программы, которая отображает графические файлы в формате GIF (ShowGif.exe). DTD также определяет пустой элемент с именем COVERIMAGE и атрибут типа ENTITY для этого элемента с именем Source.

Листинг 6.7. Определение файла Faun.gif

```
<?xml version="1.0"?>
<!DOCTYPE BOOK
[
  <!ELEMENT BOOK (TITLE, AUTHOR, COVERIMAGE)>
  <!ELEMENT TITLE (#PCDATA)>
  <!ELEMENT AUTHOR (#PCDATA)>
  <!ELEMENT COVERIMAGE EMPTY>
  <!ATTLIST COVERIMAGE Source ENTITY #REQUIRED>
  <!NOTATION GIF SYSTEM "ShowGif.exe">
  <!ENTITY faun SYSTEM "Faun.gif" NDATA GIF>
]
>
<BOOK>
  <TITLE>The Marble Faun</TITLE>
```

```
<AUTHOR>Nathaniel Hawthorne</AUTHOR>  
<COVERIMAGE Source="faun"/>  
</BOOK>
```

В элементе Документ атрибуту Source элемента COVERIMAGE присвоено имя внешнего примитива, который содержит графические данные для отображения рисунка обложки. Поскольку Source имеет тип ENTITY, вы можете присвоить ему имя общего внешнего неразбираемого примитива. Фактически единственный способ использования этого типа примитива состоит в присвоении его имени атрибуту с типом ENTITY или ENTITIES.

 *Примечание.* В отличие от файла внешнего разбираемого примитива, к файлу внешнего неразбираемого примитива XML-процессор прямого доступа не имеет. Процессор просто делает примитив и его нотацию доступными приложению, которое может выполнять необходимые действия с этой информацией.

6.3. ОБЪЯВЛЕНИЕ НОТАЦИЙ

Нотация описывает определенный формат данных. Это делается путем указания адреса описания формата, адреса программы, которая может обрабатывать данные в этом формате, либо просто описания формата. Вы можете использовать нотацию, чтобы описать формат общего внешнего неразбираемого примитива, или можете присвоить нотацию атрибуту, который имеет нумерованный тип NOTATION.

Нотация имеет следующую форму записи:

```
<!NOTATION ИмяНотации SYSTEM СистемЛитерал>
```

Здесь ИмяНотации есть имя нотации. Вы можете выбрать любое имя при условии, что оно начинается с буквы или символа подчеркивания (), после чего могут идти или не идти другие буквы, цифры, точки (.), тире (–) или символы подчеркивания. Лучше выбирать информативное имя, позволяющее идентифицировать формат. Например, если вы определяете нотацию, описывающую точечный формат (bitmap), вам следует использовать имя BMP.

СистемЛитерал – системный литерал, который может быть ограничен одинарными (') или двойными (") кавычками и содержать любые символы, за исключением символа кавычек, используемого в качестве ограничителя. Вы можете включить в системный литерал

любое описание формата, которое проинформирует приложение, как отображать или обрабатывать XML-документ. Например, вы можете включить в системный литерал одно из следующих описаний.

URI программы, которая может обрабатывать или отображать формат данных, как в следующих примерах:

```
<!NOTATION BMP SYSTEM "Pbrush.exe">  
<!NOTATION GIF SYSTEM "HYPERLINK 'http://bogus.com/ShowGif.exe'  
http://bogus.com/ShowGif.exe">
```

URI документа в сети, который описывает формат данных, например:

```
<!NOTATION STRANGIFORMAT SYSTEM  
"http://bogus.com/StrangeFormat.htm">
```

Простое описание формата, например:

```
<!ENTITY % ИмяПримитива ЗначениеПримитива>
```

Дополнительная информация об URI и примеры приведены в 5.10.1 «Использование только внешнего подмножества DTD».

6.4. ОБЪЯВЛЕНИЕ ПАРАМЕТРИЧЕСКИХ ПРИМИТИВОВ

Форма объявления разметки параметрического примитива аналогична форме объявления, используемой для общих примитивов. В последующих пунктах вы узнаете, как объявлять оба типа параметрических примитивов.

6.4.1. Объявление параметрического внутреннего разбираемого примитива

Объявление параметрического внутреннего разбираемого примитива имеет следующую общую форму записи:

```
<!ENTITY % ИмяПримитива ЗначениеПримитива>
```

Здесь ИмяПримитива есть имя примитива. Вы можете выбрать любое имя, соблюдая следующие правила:

- имя должно начинаться с буквы или символа подчеркивания (), вслед за которым могут идти или не идти буквы, цифры, точки (.), тире (–) или символы подчеркивания;

– примитив может иметь такое же имя, что и общий примитив в документе. (Параметрические и общие примитивы занимают различные пространства имен.) Имя примитива также может совпадать с именем элемента или атрибута;

– помните, что для всего текста разметки, в том числе и для имен примитивов, имеет значение регистр, в котором набраны символы. Так, примитив под именем Spot будет отличаться от примитива под именем spot.

ЗначениеПримитива есть значение примитива. Значение, присваиваемое параметрическому примитиву, представляет собой группу символов, заключенных в кавычки (литерал). Вы можете присвоить параметрическому примитиву любое значение типа литерал при условии соблюдения следующих правил:

1) строка должна быть ограничена одинарными (') или двойными (") кавычками;

2) строка не может содержать символов кавычек, которые используются в качестве ограничителей;

3) строка не может содержать символа процентов (%) и символа амперсанда (&), если это не начальный символ в ссылке на символ или общий примитив;

4) вы можете поместить параметрический примитив в DTD только как объявление разметки, но не внутри объявления разметки. Следовательно, строка ЗначениеПримитива должна содержать один или несколько типов объявлений разметки, которые разрешено использовать в DTD. Эти типы объявлений разметки описаны в 5.2.1 «Создание DTD». В частности, параметрический примитив может содержать объявления типа элемента, объявления списка атрибутов, объявления общих примитивов, объявления нотаций, инструкции по обработке или комментарии.

⇒ *Примечание.* Приведенные здесь правила использования значений примитивов, которые вы можете без опасения применять в любых ситуациях, являются несколько упрощенными в отличие от правил, содержащихся в спецификации XML. Спецификация, в определенных обстоятельствах, разрешает вам включать в значение примитива дополнительные составляющие, а также помещать ссылку на примитив внутри разметки и между объявлениями разметки.

Например, следующее DTD объявляет параметрический внутренний неразбираемый примитив с именем author, который содержит три объявления разметки: комментарий, объявление типа элемента

и объявление списка атрибутов. Содержимое примитива (т. е. замещающий его текст) вставляется в конец DTD посредством ссылки на параметрический примитив (%author;).

Листинг 6.8. Примитив с именем author

```
<!DOCTYPE BOOK
[
  <!ENTITY % author
    "<!-- Информация об авторе. -->
    <!ELEMENT AUTHOR (#PCDATA)>
    <!ATTLIST AUTHOR Nationality CDATA 'American'>"
  >
  <!ELEMENT BOOK (TITLE, AUTHOR)>
  <!ELEMENT TITLE (#PCDATA)>
  %author;
]
```

Обратите внимание, что значение атрибута по умолчанию, которое содержится в объявлении примитива ('American'), ограничено одинарными кавычками, чтобы избежать использования такого же символа, который применяется для ограничения всего значения примитива. Приведенное выше DTD эквивалентно следующему.

Листинг 6.9. Ограничение значения примитива

```
<!DOCTYPE BOOK
[
  <!ELEMENT BOOK (TITLE, AUTHOR)>
  <!ELEMENT TITLE (#PCDATA)>
  <!-- Информация об авторе. -->
  <!ELEMENT AUTHOR (#PCDATA)>
  <!ATTLIST AUTHOR Nationality CDATA 'American'>
]
```

6.4.2. Объявление параметрического внешнего разбираемого примитива

Объявление параметрического внешнего разбираемого примитива имеет следующую форму записи:

```
<!ENTITY % ИмяПримитива SYSTEM СистемЛитерал>
```

Здесь ИмяПримитива есть имя примитива. Вы можете выбрать любое имя, следуя правилам задания имен для параметрических примитивов, приведенных в предыдущем пункте.

СистемЛитерал – системный литерал, который описывает местонахождение файла, содержащего данные примитива. Системный литерал может быть заключен в одинарные (') или двойные (") кавычки и содержать любые символы, за исключением символа кавычек, который используется в качестве ограничителя.

Системный литерал задает URI файла, содержащего данные параметрического примитива. В настоящее время URI практически аналогичен стандартному унифицированному указателю ресурса Internet (URL). Вы можете использовать как полностью заданный URI, например:

```
<!ENTITY % declarations SYSTEM "HYPERLINK  
"http://bogus.com/documents/Declarations.dtd"  
http://bogus.com/documents/Declarations.dtd">
```

так и частичный URI, задающий местонахождение относительно местонахождения XML-документа, который содержит URI, например:

```
<!ENTITY % declarations SYSTEM "Declarations.dtd">
```

Относительные URI в XML-документах работают подобно относительным URL для HTML-страниц.

Файл параметрического внешнего примитива должен содержать полные объявления разметки всех типов, допустимых в DTD. В частности, он может содержать объявления типа элемента, объявления списка атрибутов, объявления примитивов, объявления нотаций, инструкции по обработке или комментарии. (Эти типы объявлений разметки описаны в 5.2.1 «Создание DTD».) Вы также можете включать ссылки на параметрические примитивы и разделы INCLUDE и IGNORE.

Вы можете использовать параметрические внешние разбираемые примитивы для хранения группы взаимосвязанных объявлений. Допустим, вы занимаетесь продажей книг, CD-ROM, плакатов и другой продукции. Вы можете поместить объявления для каждого вида продукции в отдельный файл. Это позволит вам объединять эти группы объявлений различными способами. Например, вы хотели бы создать XML-документ, который описывает только имеющиеся у вас в наличии книги и CD-ROM. Для этого вы можете поместить объявления для книг и CD-ROM в DTD документа с помощью параметрических внешних разбираемых примитивов, как показано в следующем примере XML-документа.

Листинг 6.10. Параметрические внешние разбираемые примитивы

```
<?xml version="1.0"?>
<!DOCTYPE INVENTORY
[
  <!ELEMENT INVENTORY (BOOK | CD)*>
  <!ENTITY % book_decls SYSTEM "Book.dtd">
  <!ENTITY % cd_decls SYSTEM "CD.dtd">
  %book_decls;
  %cd_decls;
]
>
<INVENTORY>
  <BOOK>
    <BOOKTITLE>The Marble Faun</BOOKTITLE>
    <AUTHOR>Nathaniel Hawthorne</AUTHOR>
    <PAGES>473</PAGES>
  </BOOK>
  <CD>
    <CDTITLE>Concerti Grossi Opus 3</CDTITLE>
    <COMPOSER>Handel</COMPOSER>
    <LENGTH>72 minutes</LENGTH>
  </CD>
  <BOOK>
    <BOOKTITLE>Leaves of Grass</BOOKTITLE>
    <AUTHOR>Walt Whitman</AUTHOR>
    <PAGES>462</PAGES>
  </BOOK>
  <!-- Дополнительные виды продукции... -->
</INVENTORY>
```

Ниже показано содержимое файла примитива Book.dtd.

Листинг 6.11. Содержимое файла примитива Book.dtd

```
<!ELEMENT BOOK (BOOKTITLE, AUTHOR, PAGES)>
<!ELEMENT BOOKTITLE (#PCDATA)>
<!ELEMENT AUTHOR (#PCDATA)>
<!ELEMENT PAGES (#PCDATA)>
```

А вот содержимое файла примитива CD.dtd.

Листинг 6.12. Содержимое файла примитива CD.dtd

```
<!ELEMENT CD (CDTITLE, COMPOSER, LENGTH)>
<!ELEMENT CDTITLE (#PCDATA)>
<!ELEMENT COMPOSER (#PCDATA)>
<!ELEMENT LENGTH (#PCDATA)>
```

Заметим, что параметрический внешний разбираемый примитив работает во многом аналогично внешнему подмножеству DTD. Параметрические внешние примитивы, однако, обеспечивают

большую гибкость – они разрешают вам включать несколько файлов внешних объявлений, причем в любом порядке. (Напомним, что внешнее подмножество DTD всегда обрабатывается после того, как полностью будет обработано подмножество внутреннего DTD.)

6.5. ВСТАВКА ССЫЛОК НА ПРИМИТИВ

Вставка содержимого (замещающий текст) примитива в документ осуществляется с помощью ссылок на примитив. Вы уже познакомились с несколькими примерами ссылок на примитив. Общий вид ссылки на общий примитив следующий:

`&ИмяПримитива;`

а на параметрический примитив:

`%ИмяПримитива;`

где `ИмяПримитива` есть имя, присваиваемое примитиву в объявлении. Исключением является общий внешний неразбираемый примитив, который вы не можете вставить с использованием ссылки. Единственный способ использования этого типа примитива заключается в присвоении его имени атрибуту, имеющему тип ENTITY или ENTITIES.

Объявление примитива должно предшествовать любой ссылке на этот примитив.

Для каждого типа примитива в представленной ниже табл. 6.1 приведена форма записи ссылки на примитив и перечень возможных мест помещения ссылки на примитив.

Таблица 6.1

Формы записи ссылок на примитив и символ

Виды ссылок на примитивы и символ	Форма записи ссылки	Место, в которое вы можете поместить ссылку на примитив и символ (пример)
1	2	3
Общий внутренний разбираемый примитив	<code>&ИмяПримитива;</code>	В содержимое элемента. В значение атрибута (как значение по умолчанию в объявлении атрибута либо в начальном теге элемента) (см. п. 6.2.1). В значение в объявлении внутреннего примитива (см. п. 6.2.1)

1	2	3
Общий внешний разбираемый примитив	&ИмяПримитива;	В содержимое элемента (см. п. 6.2.2)
Общий внешний неразбираемый примитив	АтрПрим= 'ИмяПримитива' где АтрПрим есть атрибут типа ENTITY или ENTITIES	В значение в объявлении внутреннего примитива (см. п. 6.2.3). Вы не можете поместить ссылку на этот тип примитива, но можете присвоить имя примитива атрибуту, имеющему тип ENTITY или ENTITIES (см. п. 6.2.3)
Параметрический внутренний разбираемый примитив	#ИмяПримитива	В DTD в место помещения объявлений разметки (см. п. 6.4.1). Исключения приведены в разделе 4 спецификации XML, доступ по адресу http://www.w3.org/TR/REC-xml
Символ		 или &#xh; где 9 – десятичный числовой код символа, а h – шестнадцатеричный числовой код	В содержимое элемента (см. подгл. 6.6). В значение атрибута (в качестве значения по умолчанию в объявлении списка атрибутов или в начальном теге элемента) (см. подгл. 6.6). В значение в объявлении внутреннего примитива (см. подгл. 6.6)

В табл. 6.1 также приведены ссылки на пункты и подглавы, в которых вы можете найти примеры. О ссылках на символы речь пойдет в следующей подглаве, но данный вид ссылок также включен в табл. 6.1.

Примеры ссылок на примитив:

– в следующем XML-документе объявлены два общих внутренних разбираемых примитива (am и en). Документ использует ссылку на am для присвоения значения по умолчанию атрибуту Nationality и ссылку на en для присвоения значения атрибуту Nationality элемента AUTHOR. Преимущество использования примитива здесь заключается в том, что вы можете изменить значение по всему документу простым редактированием определения примитива (например, изменив значение en с «English» на «British»).

Листинг 6.13. Ссылки на примитив

```
<!DOCTYPE INVENTORY
[
  <!ENTITY am "American">
  <!ENTITY en "English">
  <!ELEMENT INVENTORY (BOOK*)>
  <!ELEMENT BOOK (TITLE, AUTHOR)>
  <!ELEMENT TITLE (#PCDATA)>
  <!ELEMENT AUTHOR (#PCDATA)>
  <!ATTLIST AUTHOR Nationality CDATA "&am;">
]
>
<INVENTORY>
<BOOK>
  <TITLE>David Copperfield</TITLE>
  <AUTHOR Nationality="&en;">Charles Dickens</AUTHOR>
</BOOK>
<!-- Другие элементы... -->
</INVENTORY>
```

– следующее DTD определяет общий внутренний разбираемый примитив (int_entity) и общий внешний разбираемый примитив (ext_entity). Затем он определяет другой общий внутренний разбираемый примитив (combo_entity) и помещает оба предыдущих примитива в значение примитива combo_entity.

Листинг 6.14. Примитивы int_entity, ext_entity, combo_entity

```
<!DOCTYPE INVENTORY
[
  <!ENTITY int_entity "internal entity value">
  <!ENTITY ext_entity SYSTEM "Entity.xml">
  <!ENTITY combo_entity
    "значение, состоящее из &ext_entity; плюс
    &int_entity;">
  <!-- Другие объявления разметки... -->
]
>
```

6.6. ВСТАВКА ССЫЛОК НА СИМВОЛЫ

Вы можете воспользоваться ссылкой на символ, чтобы вставить символ, которого нет на вашей клавиатуре (например, ä), либо вставить символ, который не допускает его вставку в данном контексте как литерал (например, символы < или & в составе символьных

данных элемента). Вам не нужно делать какие-либо определения перед использованием ссылки на символ – вы можете просто поместить ее в нужном месте.

Ссылка на символ имеет две различные формы. Первая форма:

`	`

где 9 – это одна или несколько десятичных цифр (от 0 до 9), представляющих числовой код символа в наборе символов ISO/IEC 10646.

Вторая форма ссылки на символ:

`&#xh;`

где h – это одна или несколько шестнадцатеричных цифр (от 0 до F), также представляющих числовой код символа в наборе ISO/IEC 10646.

Например, обе ссылки `A` и `A` означают вставку прописной буквы A.

Международная таблица символов ISO/IEC 10646 используется для числового представления символов практически любых языков. (ISO – International Organization for Standardization, IEC – International Electrotechnical Commission.)

⇒ *Примечание.* Список мест, в которые вы можете поместить ссылку на символ в вашем документе, приведен в табл. 6.1 (см. на с. 115–116). Там же указаны и примеры.

Для следующего элемента символ левой угловой скобки (<) вставляется в символьные данные элемента с помощью ссылки на символ `<` (60 есть десятичный код для <). Напомним, что непосредственная вставка символа < в символьные данные не допускается:

`<TITLE><The Legend of Sleepy Hollow></TITLE>`

Для нижеприведенного элемента ссылка на символ `ä` используется для вставки символа ä в значение атрибута:

`<RESIDENT Address="Seilerstätte 30, Wien">Mike Young</RESIDENT>`

В следующем объявлении общего внутреннего разбираемого примитива в DTD ссылка на символ `%` применяется для вставки символа процентов (%) (37 есть десятичный код для %), который не может быть непосредственно введен как литерал в значение внутреннего примитива:

`<!ENTITY heading1 "% Complete">`

6.7. ИСПОЛЬЗОВАНИЕ ПРЕДВАРИТЕЛЬНО ОПРЕДЕЛЕННЫХ ПРИМИТИВОВ

В XML-документе вы можете использовать ссылку на заранее определенный примитив, чтобы вставить следующие пять стандартных символов в места, куда вставка символов как литералов не допускается (табл. 6.2).

Таблица 6.2

Ссылки на предопределенный примитив

Ссылка на предопределенный примитив	Вставляемый символ	Эквивалентная ссылка на символ
&	&	&
<	<	<
>	>	>
'	'	'
"	"	"

Вставка этих ссылок на предопределенный примитив эквивалентна вставке соответствующей ссылки на символ. Ссылки на предопределенные примитивы легче запомнить, а документ при этом легче воспринимается.

Эти предопределенные примитивы похожи на другие общие внутренние разбираемые примитивы, за исключением того, что ссылки на них вы можете использовать без определения примитивов и вы можете вставлять их в те же места, что и примитивы данного типа, а именно:

- 1) в содержимое элемента;
- 2) в значение атрибута (как значение по умолчанию в объявлении атрибута или в начальном теге элемента);
- 3) в значение в объявлении внутреннего примитива.

В следующих трех примерах ссылки на предопределенные примитивы используются для вставки символов, которые не допускаются вставлять как литералы.

В первом примере ссылка < применяется для вставки в содержимое элемента левой угловой скобки (<):

```
<TITLE>&lt;The Legend of Sleepy Hollow></TITLE>
```

Во втором примере ссылка & используется для вставки амперсанда (&) в значение атрибута:

```
<PRODUCT Company="Ongaro &amp; Sons">3/4" T fitting</PRODUCT>
```

В третьем примере ссылка применяется для вставки двойных кавычек (") в значение примитива (их нельзя вставить как литерал, поскольку такие же кавычки используются как ограничители строки):

```
<!ENTITY heading "Christopher &quot;Kit&quot;; Carson">
```

6.8. ОБЪЯВЛЕНИЕ ДОКУМЕНТА АВТОНОМНЫМ (STANDALONE)

Вы можете включить объявление автономности (standalone) документа в XML-объявление. Объявление standalone сообщает процессору, требуются ли внешние объявления для обработки документа.

Если XML-документ имеет внешние объявления разметки (либо во внешнем подмножестве DTD, либо в параметрическом внешнем разбираемом примитиве), но ни одно из этих объявлений не оказывает влияния на содержимое документа, передаваемого XML-процессором приложению, вы можете установить для standalone значение 'yes' или "yes", как в следующем XML-объявлении:

```
<?xml version="1.0" standalone="yes"?>
```

Установка предотвращает ненужную обработку внешних файлов. Установка для standalone значения 'no' или "no" либо пропуск объявления standalone сообщает процессору, что он должен обработать все внешние объявления разметки, поскольку одно или несколько из них оказывают влияние на содержимое документа.

ЗАДАНИЯ

Добавьте примитивы в документ

В вашем текстовом редакторе откройте документ Inventory Valid.xml, созданный вами в главе 5.

В начало DTD документа (блок текста, ограниченный символами [] в верхней части документа) добавьте следующие объявления примитива и нотации.

Листинг 6.15. Объявление примитива и нотации

```
<!-- Примитивы для присвоения значений элементу BINDING: -->  
<!ENTITY mass "mass market paperback">
```



```

<!ENTITY trade "trade paperback">
<!ENTITY hard "hardcover">
<!-- Внешние примитивы, содержащие обзоры. -->
<!-- Они будут присвоены атрибуту Review элементов BOOK. -->
<!NOTATION DOC SYSTEM "Microsoft Word document">
<!NOTATION TXT SYSTEM "plain text file">
<!ENTITY rev_leaves
  SYSTEM "Review Leaves of Grass.doc"
  NDATA DOC>
<!ENTITY rev_faun1
  SYSTEM "Review 01 of The Marble Faun.doc"
  NDATA DOC>
<!ENTITY rev_faun2
  SYSTEM "Review 02 of The Marble Faun.txt"
  NDATA TXT>
<!ENTITY rev_screw
  SYSTEM "Review of The Turn of the Screw.txt"
  NDATA TXT>

```

Первые три примитива представляют собой внутренние разбираемые примитивы, которые вы можете вставлять в элементы BINDING вместо того, чтобы вводить описание типа переплета для каждого элемента. Использование примитивов обеспечивает однозначность ваших описаний типов переплета для различных книг. Кроме того, примитивы облегчают модификацию описания. (Например, вы можете заменить «hardcover» на «hardback» для каждого элемента BINDING, в котором встречается этот тип обложки, простым редактированием примитива hard.)

Следующие (и последние) четыре примитива представляют собой общие внешние неразбираемые примитивы, которые позволяют вам подключать внешние файлы, содержащие обзоры книг для элементов BOOK.

Добавьте атрибут Reviews в объявление списка атрибутов для элемента BOOK в DTD следующим образом:

```

<!ATTLIST BOOK      InStock      (yes | no)      #REQUIRED
Reviews ENTITIES #IMPLIED>

```

Reviews представляет собой необязательный атрибут (#IMPLIED), которому вы можете присвоить имена одного или нескольких общих внешних неразбираемых примитивов (Reviews имеет тип ENTITIES).

В каждом элементе BINDING замените описание переплета соответствующей ссылкой на примитив. Например, вы должны изменить элемент BINDING для книги The Adventures of Huckleberry Finn

```
<BINDING>mass market paperback</BINDING>
```

на

```
<BINDING>&mass;</BINDING>
```

Добавьте атрибуты Reviews элементам BOOK следующим образом:

- для Leaves of Grass: <BOOK InStock="no" Reviews="rev_leaves">
- для The Marble Faun: <BOOK InStock="yes" Reviews="rev_faun1 rev_faun2">
- для The Turn of the Screw: <BOOK InStock="no" Reviews="rev_screw">

Чтобы отразить новое имя файла, которое вы собираетесь присвоить, измените комментарий в начале документа

```
<!-- Имя файла: Inventory Valid.xml -->
```

на

```
<!-- Имя файла: Inventory Valid Entity.xml -->
```

Воспользуйтесь командой Save As (Сохранить как) вашего текстового редактора, чтобы сохранить копию модифицированного документа под именем Inventory Valid Entity.xml.

Полный XML-документ представлен ниже.

Листинг 6.16. Inventory Valid Entity.xml

```
<?xml version="1.0" encoding="windows-1251"?>
<!-- Имя файла: Inventory Valid Entity.xml -->
<!DOCTYPE INVENTORY
[
  <!-- Примитивы для присвоения значений элементу BINDING: -->
  <!ENTITY mass "mass market paperback">
  <!ENTITY trade "trade paperback">
  <!ENTITY hard "hardcover">
  <!-- Внешние примитивы, содержащие обзоры. -->
  <!-- Они будут присвоены атрибуту Review элементов BOOK. -->
  <!NOTATION DOC SYSTEM "Microsoft Word document">
  <!NOTATION TXT SYSTEM "plain text file">
  <!ENTITY rev_leaves SYSTEM "Review of Leaves of Grass.doc"
NDATA DOC>
  <!ENTITY rev_faun1 SYSTEM "Review 01 of The Marble
Faun.doc" NDATA DOC>
```

```

        <!ENTITY rev_faun2 SYSTEM "Review 02 of The Marble
Faun.txt" NDATA TXT>
        <!ENTITY rev_screw SYSTEM "Review of The Turn of
the Screw.txt" NDATA TXT>
        <!ELEMENT INVENTORY (BOOK)*>
        <!ELEMENT BOOK (TITLE, AUTHOR, BINDING, PAGES, PRICE)>
        <!ATTLIST BOOK InStock (yes|no) #REQUIRED Reviews
ENTITIES #IMPLIED>
        <!ELEMENT TITLE (#PCDATA | SUBTITLE)*>
        <!ELEMENT SUBTITLE (#PCDATA)>
        <!ELEMENT AUTHOR (#PCDATA)>
        <!ATTLIST AUTHOR Born CDATA #IMPLIED>
        <!ELEMENT BINDING (#PCDATA)>
        <!ELEMENT PAGES (#PCDATA)>
        <!ELEMENT PRICE (#PCDATA)>
    ]
>
<INVENTORY>
    <BOOK InStock="yes">
        <TITLE>The      Adventures      of      Huckleberry
Finn</TITLE>
        <AUTHOR Born="1835">Mark Twain</AUTHOR>
        <BINDING>&mass;</BINDING>
        <PAGES>298</PAGES>
        <PRICE>$5.49</PRICE>
    </BOOK>
    <BOOK InStock="no" Reviews="rev_leaves">
        <TITLE>Leaves of Grass</TITLE>
        <AUTHOR Born="1819">Walt Whitman</AUTHOR>
        <BINDING>&hard;</BINDING>
        <PAGES>462</PAGES>
        <PRICE>$7.75</PRICE>
    </BOOK>
    <BOOK InStock="yes">
        <TITLE>The Legend of Sleepy Hollow</TITLE>
        <AUTHOR>Washington Irving</AUTHOR>
        <BINDING>&mass;</BINDING>
        <PAGES>98</PAGES>
        <PRICE>$2.95</PRICE>
    </BOOK>
    <BOOK InStock="yes" Reviews="rev_faun1 rev_faun2">
        <TITLE>The Marble Faun</TITLE>
        <AUTHOR Born="1804">Nathaniel Hawthorne</AUTHOR>
        <BINDING>&trade;</BINDING>
        <PAGES>473</PAGES>
        <PRICE>$10.95</PRICE>

```

```

</BOOK>
<BOOK InStock="no">
  <TITLE>Moby-Dick
  <SUBTITLE>Or, the Whale</SUBTITLE>
</TITLE>
  <AUTHOR Born="1819">Herman Melville</AUTHOR>
  <BINDING>&hard;</BINDING>
  <PAGES>724</PAGES>
  <PRICE>$9.95</PRICE>
</BOOK>
<BOOK InStock="yes">
  <TITLE>The Portrait of a Lady</TITLE>
  <AUTHOR>Henry James</AUTHOR>
  <BINDING>&mass;</BINDING>
  <PAGES>256</PAGES>
  <PRICE>$4.95</PRICE>
</BOOK>
<BOOK InStock="yes">
  <TITLE>The Scarlet Letter</TITLE>
  <AUTHOR>Nathaniel Hawthorne</AUTHOR>
  <BINDING>&trade;</BINDING>
  <PAGES>253</PAGES>
  <PRICE>$4.25</PRICE>
</BOOK>
<BOOK InStock="no" Reviews="rev_screw">
  <TITLE>The Turn of the Screw</TITLE>
  <AUTHOR>Henry James</AUTHOR>
  <BINDING>&trade;</BINDING>
  <PAGES>384</PAGES>
  <PRICE>$3.35</PRICE>
</BOOK>
</INVENTORY>

```

Если вы хотите проверить валидность вашего документа, воспользуйтесь сценарием проверки на валидность, приведенным в пункте «Проверка валидности XML-документа» (см. на с. 291) в главе 9.

Глава 7. ТАБЛИЦА СТИЛЕЙ

7.1. ОТОБРАЖЕНИЕ XML-ДОКУМЕНТОВ С ИСПОЛЬЗОВАНИЕМ ТАБЛИЦ КАСКАДНЫХ СТИЛЕЙ

В этой главе вы познакомитесь с первым из методов отображения XML-документов в браузере Microsoft Internet Explorer 5, рассматриваемых в этом курсе, – таблицами каскадных стилей (CSS). Таблица каскадных стилей представляет собой файл, который содержит инструкции для форматирования элементов в XML-документе.

Поскольку в XML вы создаете свои собственные элементы, браузер не имеет встроенных средств, позволяющих определить, как их правильно отобразить. Создание таблицы каскадных стилей и связывание ее с вашим XML-документом – это один из способов сообщить браузеру, как отображать каждый из элементов документа. XML-документ со связанной таблицей каскадных стилей может быть открыт непосредственно в Internet Explorer 5. Вам нет необходимости использовать HTML-страницу для доступа и отображения данных.

Хранение инструкций по отображению в таблице стилей отдельно от самого XML-документа повышает гибкость XML-документа и облегчает работу с ним. Вы можете, например, быстро адаптировать один XML-документ к различным условиям отображения простым присоединением соответствующей таблицы стилей, без необходимости реструктурировать сам документ. Вы также можете быстро обновить формат для группы сходных XML-документов с помощью внесения изменений в присоединенной к этим документам таблице стилей, не открывая и не редактируя каждый из документов.

Использование таблицы стилей, наверное, является самым простым методом отображения XML-документа. Язык CSS уже знаком многим Web-дизайнерам, поскольку именно он в настоящее время используется для HTML-страниц. Кроме того, современные Web-браузеры обеспечивают высокий уровень поддержки таблиц каскадных стилей, в то время как другие методы отображения XML все еще находятся в стадии развития, и браузеры только начинают их поддерживать.

Тем не менее по сравнению с другими методами отображения XML, о которых вы узнаете в последующих главах, таблицы каскадных стилей имеют ряд ограничений. Хотя таблица каскадных стилей предоставляет достаточно высокий уровень управляемости способами, которыми браузер форматирует содержимое элементов в XML-документах, она не дает возможности модифицировать или реорганизовывать его содержимое. Она также не позволяет вам осуществлять доступ к атрибутам, примитивам, инструкциям по обработке и другим компонентам XML, а также не дает возможности обрабатывать информацию, которую эти компоненты содержат.

В последующих главах вы познакомитесь с более сложными, но и более гибкими способами отображения XML-документов. В главе 8 вы узнаете, как связывать XML-документ с HTML-страницей и отображать XML-элементы путем сцепления с ними стандартных HTML-элементов.



Примечание. В этой главе раскрывается большинство свойств CSS, поддерживаемых Internet Explorer 5, которые являются частью оригинальной версии CSS, установленной консорциумом World Wide Web (W3C) и известной как Cascading Style Sheets Level 1, или CSS1. Консорциум W3C также определил усовершенствованную версию CSS, которая значительно превосходит версию CSS1 и известна как Cascading Style Sheets Level 2, или CSS2. Версия CSS2 только частично поддерживается современными браузерами и не рассматривается в этом курсе.

7.2. ОСНОВНЫЕ ЭТАПЫ ПРИ ИСПОЛЬЗОВАНИИ ТАБЛИЦЫ КАСКАДНЫХ СТИЛЕЙ

Вот два основных этапа при использовании таблицы каскадных стилей для отображения XML-документа:

- 1) создание файла таблицы стилей;
- 2) связывание таблицы стилей с XML-документом.

Таблица каскадных стилей представляет собой текстовый файл, обычно с расширением .css, который содержит набор правил, сообщающих браузеру, каким образом форматировать и отображать элементы в определенном XML-документе. Как и в XML-документе, вы можете создавать таблицу стилей с помощью вашего любимого текстового редактора. Листинг 7.1 содержит пример простой таблицы каскадных стилей.

Листинг 7.1. Inventory01.css

```
/* File Name: Inventory01.css */
BOOK
    {display:block;
    margin-top:12pt;
    font-size:10pt}
TITLE
    {font-style:italic}
AUTHOR
    {font-weight:bold}
```

Эта таблица стилей предназначена для присоединения к XML-документу, представленному в листинге 7.2, который используется и в других примерах в данной главе, поэтому вам придется не раз к нему обращаться.

Листинг 7.2. Inventory01.xml

```
<?xml version="1.0"?>
<!-- File Name: Inventory01.xml -->
<?xml-stylesheet type="text/css" href="Inventory01.css"?>
<INVENTORY>
    <BOOK>
        <TITLE>The Adventures of Huckleberry Finn</TITLE>
        <AUTHOR>Mark Twain</AUTHOR>
        <BINDING>mass market paperback</BINDING>
        <PAGES>298</PAGES>
        <PRICE>$5.49</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>Leaves of Grass</TITLE>
        <AUTHOR>Walt Whitman</AUTHOR>
        <BINDING>hardcover</BINDING>
        <PAGES>462</PAGES>
        <PRICE>$7.75</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>The Legend of Sleepy Hollow</TITLE>
        <AUTHOR>Washington Irving</AUTHOR>
        <BINDING>mass market paperback</BINDING>
        <PAGES>98</PAGES>
        <PRICE>$2.95</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>The Marble Faun</TITLE>
        <AUTHOR>Nathaniel Hawthorne</AUTHOR>
        <BINDING>trade paperback</BINDING>
```

```

        <PAGES>473</PAGES>
        <PRICE>$10.95</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>Moby-Dick</TITLE>
        <AUTHOR>Herman Melville</AUTHOR>
        <BINDING>hardcover</BINDING>
        <PAGES>724</PAGES>
        <PRICE>$9.95</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>The Portrait of a Lady</TITLE>
        <AUTHOR>Henry James</AUTHOR>
        <BINDING>mass market paperback</BINDING>
        <PAGES>256</PAGES>
        <PRICE>$4.95</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>The Scarlet Letter</TITLE>
        <AUTHOR>Nathaniel Hawthorne</AUTHOR>
        <BINDING>trade paperback</BINDING>
        <PAGES>253</PAGES>
        <PRICE>$4.25</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>The Turn of the Screw</TITLE>
        <AUTHOR>Henry James</AUTHOR>
        <BINDING>trade paperback</BINDING>
        <PAGES>384</PAGES>
        <PRICE>$3.35</PRICE>
    </BOOK>
</INVENTORY>

```

⇒ *Примечание.* Пример таблицы стилей в листинге 7.1 и пример XML-документа в листинге 7.2 являются копиями файлов, которые вы создали в упражнении «Отобразите XML-документ с использованием таблицы каскадных стилей» в главе 2.

Таблица стилей состоит из одного или нескольких правил. Правило содержит информацию по отображению определенного типа элемента в XML-документе. На рис. 7.1 представлено правило для элементов `BOOK` с указанием его составных частей.

Селектор представляет собой имя типа элемента, к которому относится информация по отображению.

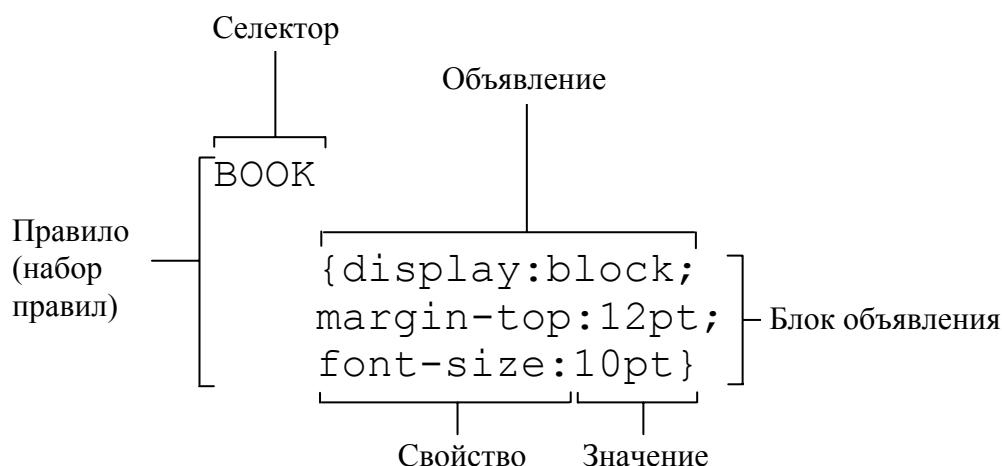


Рис. 7.1. Правило для элементов BOOK

За селектором следует блок объявлений, который ограничивается фигурными скобками ({}) и содержит одно или несколько объявлений, разделяемых точкой с запятой.

Каждое объявление задает установку определенного свойства, такого как размер шрифта, который будет использован для отображения элемента. Объявление состоит из свойства, вслед за которым идет двоеточие, после которого следует значение для данного свойства. Например, следующее объявление устанавливает для свойства font-size (размер шрифта) значение 10 pt (10 пт) (см. рис. 7.2).

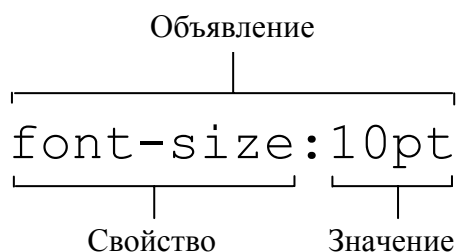


Рис. 7.2. Установка определенного свойства

Таблица стилей может также содержать комментарии. Комментарии в таблице стилей начинаются с символов косой черты и звездочки (/*) и заканчиваются символами звездочки и косой черты (* /). Между этими парами символов-ограничителей вы можете поместить любой текст по вашему желанию. Когда браузер прочитывает таблицу стилей для форматирования документа, он игнорирует этот текст. Вы можете использовать комментарий

для пояснений, указания назначения и действия таблицы стилей. В качестве примера можно привести комментарий в начале таблицы стилей из листинга 7.1:

```
/* Имя файла: Inventory01.css */
```

Вы также можете применять комментарии в процессе разработки таблицы стилей, чтобы временно отключить правило или его часть. Например, если вы хотите посмотреть, как будут выглядеть элементы BOOK без верхней отбивки, то можете временно добавить символы комментариев в следующее правило:

```
BOOK
{display:block;
/* margin-top:12pt; */
font-size:10pt}
```



Рис. 7.3. Отображение в Internet Explorer 5 страницы с использованием CSS-таблиц



Примечание. Пустые символы (пробелы, табуляция, пропуск строки) отделяют различные компоненты CSS, такие как индивидуальные объявления в блоке объявлений. Способ использования пропусков в данном курсе является лишь одной из возможностей. Вы можете использовать пропуски любым способом, чтобы лучше организовать содержимое и придать большую ясность вашим собственным таблицам стилей. Например, вы можете поместить все объявления, относящиеся к правилу, в одну строку вместо того, чтобы размещать каждое из них на отдельной строке. Пример таблицы стилей, представленной в листинге 7.1 (см. на с. 127), содержит следующие объявления:

- `display:block`. Помещает пустую строку перед и после текста элемента;
- `margin-top:12pt`. Добавляет верхнее поле (отбивку) высотой 12 пт к тексту элемента;
- `font-size:10pt`. Устанавливает размер шрифта, используемого для отображения текста элемента, в 10 пт;
- `font-style:italic`. Отображает текст элемента курсивом;
- `font-weight:bold`. Отображает текст элемента полужирным.

На рис. 7.3 показано, как Internet Explorer 5 отображает XML-документ, который использует данную таблицу стилей в соответствии с инструкциями, содержащимися в этих объявлениях.

Набор свойств, используемых в таблицах каскадных стилей, похож на набор свойств, которые вы можете применять в текстовом процессоре. Далее в этой главе вы познакомитесь с различными свойствами, которые можете использовать, а также со значениями, которые вы можете присваивать определенным свойствам.

7.2.1. Нечувствительность к регистру в CSS

В Internet Explorer 5 таблицы каскадных стилей являются нечувствительными к регистру, в котором набраны символы. Иными словами, когда Internet Explorer 5 обрабатывает таблицу стилей, он игнорирует регистр букв (строчные или прописные). Например, вы можете набрать следующее правило любым из трех приведенных ниже способов:

```
title
  {font-style:italic}
Title
  {FONT-STYLE:Italic}
title
  {Font-Style:ITALIC}
```

Нечувствительность к регистру в таблицах каскадных стилей имеет важное значение. Поскольку XML-документы являются чувствительными к регистру, вы вполне можете иметь два различных типа элемента, имена которых отличаются лишь регистром букв, например Book и BOOK. В таблице каскадных стилей, однако, эти два имени будут отнесены к одному и тому же типу элемента, и вы не сможете назначить им различную установку свойств. Следовательно, если вы предполагаете отображать ваш XML-документ с использованием таблицы каскадных стилей, у вас не должно быть типов элементов, имена которых отличаются только регистром одной или нескольких букв.

7.2.2. Наследование установок свойств

Набор свойств, которые вы присвоили определенному элементу (например, BOOK), действует на все дочерние элементы, прямо или косвенно вложенные в него, если только они не переустанавливаются впоследствии для определенного дочернего элемента.

Следующие свойства, однако, являются исключениями и не наследуются дочерними элементами:

1) свойство display, о котором пойдет речь в 7.2.9 «Установка свойства display»;

2) свойства, относящиеся к фону (background-color, background-image, background-repeat и background-position), рассматриваются в 7.5 «Установка свойств фона»;

3) свойство vertical-align, описанное в 7.6 «Установка свойств разбивки текста и выравнивания»;

4) свойства размещения текста, о которых вы узнаете в 7.7 «Установка свойств текстовых областей».

Например, таблица стилей в листинге 7.1 (см. на с. 127) форматирует элемент BOOK (документ приведен в листинге 7.2 (см. на с. 127–128)) следующим образом:

```
BOOK
{display:block;
margin-top:12pt;
font-size:10pt}
```

Каждый элемент BOOK имеет пять дочерних элементов. Поскольку свойство font-size является наследуемым, все дочерние элементы внутри элемента BOOK отображаются с размером шрифта 10 пт. Дочерние элементы, однако, не наследуют установку свойств

display и margin-top (свойство margin-top относится к группе свойств размещения текста).

Для ненаследуемых свойств, если вы не задали значение свойства для конкретного элемента, браузер использует значение свойства по умолчанию. Например, значением по умолчанию для свойства display будет inline. В этой главе приведены значения свойств по умолчанию для всех ненаследуемых свойств.

Поскольку большинство значений свойств являются наследуемыми, при разработке таблицы стилей вам лучше начать с элементов верхнего уровня, а затем опускаться к более глубоко вложенным элементам. При этом вам придется вносить минимальное количество изменений и уточнений в установку свойств (так, вам нет необходимости устанавливать свойства дочерних элементов, если унаследованные ими значения свойств вас устраивают).

Подробнее о наследовании и о его роли в механизме каскадного присвоения значения вы узнаете в 7.2.8 «Присвоение значений в таблицах каскадных стилей».

7.2.3. Использование множественных элементов и множественных правил

Вы можете применить одно правило к нескольким элементам, включив все имена элементов в селектор и отделив имена запятыми. Например, следующее правило применяется к типам элементов POEM, TITLE, AUTHOR, DATE и STANZA:

```
POEM, TITLE, AUTHOR, DATE, STANZA
{display:block;
margin-bottom:12pt}
```

Если для группы элементов устанавливается общий набор свойств, вы можете сделать вашу таблицу стилей короче и облегчить ее восприятие, включив все эти элементы в одно правило, вместо того, чтобы дублировать установки в отдельных правилах.

Вы также можете включить определенный тип элемента в более чем одно правило внутри той же самой таблицы стилей. Например, следующие оба правила включают элемент DATE:

```
POEM, TITLE, AUTHOR, DATE, STANZA
{display:block;
margin-bottom:12pt}
```

```
DATE
    {font-style:italic}
```

Первое правило содержит объявление, которое элемент DATE разделяет с другими элементами в списке, в то время как второе правило осуществляет дополнительную настройку для элемента DATE, а именно: задает установку свойства, применимого только к этому элементу.

7.2.4. Использование контекстуальных селекторов

В селекторе вы можете предварить имя элемента именами одного или нескольких элементов-предков (родительский, родительский плюс родительский родителя и т. д.), и правило будет применено только к элементам с этим именем, которые являются вложенными подобным образом. Селектор, который включает один или несколько элементов-предков, называется контекстуальным (contextual). Селектор, который не включает имен элементов-предков (подобно тем, с которыми вы имели дело в предыдущем пункте), называется родовым (generic).

Если определенное свойство для одного и того же элемента имеет одну установку в правиле с контекстуальным селектором и другую установку в правиле с родовым селектором, установка в правиле с контекстуальным селектором доминирует, поскольку является более конкретизированной.


Предположим, что следующий элемент является корневым элементом XML-документа:

```
<MAPS>
  <CITY>
    <NAME>Santa Fe</NAME>
    <STATE>New Mexico</STATE>
  </CITY>
  <STATE>California</STATE>
</MAPS>
```

Следующие правила в присоединяемой таблице стилей заставят браузер отформатировать «New Mexico» обычным шрифтом, а «California» – курсивом:

```
CITY STATE
    {font-style:normal}
STATE
    {font-style:italic}
```

Хотя содержимое «New Mexico» элемента STATE отвечает общим контекстуальным селекторам в правиле CITY STATE и родовому селектору в правиле STATE, селектор CITY STATE является более конкретным и, следовательно, имеет приоритет.

 *Примечание.* Имейте в виду, что не следует помещать запятые между именами элементов в контекстуальном селекторе. В противном случае правило будет применено ко всем элементам, но не к последнему дочернему элементу в списке.

7.2.5. Использование атрибута STYLE

Вы можете использовать специальный атрибут STYLE в вашем XML-документе вместо того, чтобы устанавливать одно или несколько определенных свойств отдельного элемента в таблице стилей. Если значение свойства, установленного с помощью атрибута STYLE, конфликтует со значением свойства, установленного в таблице стилей, установка с помощью атрибута STYLE имеет приоритет. Таким образом, атрибут STYLE является удобным средством, чтобы переустановить – для определенного элемента – значение свойства, присвоенное для типа элемента в присоединенной таблице стилей. Тем не менее использование атрибута STYLE нарушает принцип CSS относительно хранения информации о форматировании отдельно от определения содержимого документа и структуры XML-файла.

Чтобы установить одно или несколько значений свойств, включите объявления в значение атрибута STYLE в виде строки, заключенной в кавычки, отделяя индивидуальные объявления точкой с запятой, как вы это делаете в объявлении блока в CSS.

Например, таблица стилей в листинге 7.1 (см. на с. 127) задает для элементов TITLE курсивное начертание с размером шрифта 10 пт. Однако если вы включите следующий атрибут STYLE в начальный тег определенного элемента TITLE в документе, этот элемент будет отображен шрифтом roman (не курсивом), а размер шрифта составит 14 пт:

```
<TITLE STYLE='font-style:normal; font-size:14pt'>
  The Adventures of Huckleberry Finn
</TITLE>
```

В Internet Explorer 5 ваш документ будет выглядеть, как показано на рис. 7.4.

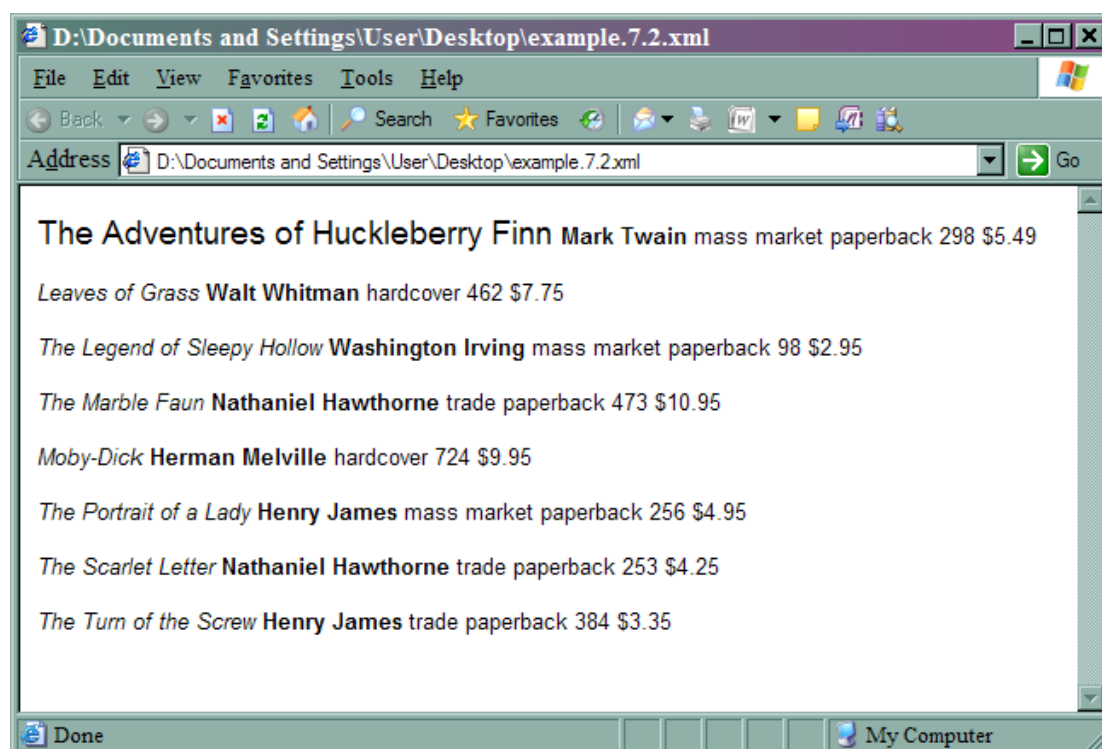


Рис. 7.4. Отображение в Internet Explorer 5 страницы с использованием атрибута STYLE



Совет. Для валидных документов вам необходимо объявить атрибут STYLE в DTD перед тем, как вы сможете использовать атрибут. Вы можете сделать это, например, следующим образом:

```
<!ATTLIST TITLE STYLE CDATA #IMPLIED>
```

7.2.6. Импорт других таблиц стилей

Вы можете воспользоваться директивой `@import` в вашей таблице каскадных стилей, чтобы встроить в нее одну или несколько других таблиц стилей. Возможность импорта отдельных таблиц стилей позволяет вам хранить правила для связанных стилей в отдельных файлах, а затем объединять их при создании документов определенного типа.

Вот обобщенная форма записи директивы `@import`, где URLТаблСтил есть полный или относительный URL (Uniform Resource Locator) файла, содержащего таблицу каскадных стилей, которую вы хотите импортировать:

```
@import url(URLТаблСтил);
```


Сведения о задании значений URL приведены далее в 7.2.7 «Задание значений URL». Например, следующая директива (использующая относительный URL), помещенная в начале таблицы стилей в листинге 7.1 (см. на с. 127), импортирует таблицу стилей, содержащуюся в файле Book.css (который должен находиться в той же папке, что и основная таблица стилей):

```
/* Имя файла: Inventory01.css */
@import url (Book.css);
BOOK
    {display:block;
    margin-top:12pt;
    font-size:10pt}
/* продолжение таблицы стилей... */
```

Директива @import должна располагаться в начале таблицы стилей перед правилами. Вы можете поместить в начале таблицы стилей несколько директив @import.

Если вы импортируете одну или несколько таблиц стилей, браузер объединяет правила, содержащиеся в основной и импортируемых таблицах стилей. Однако в случае возникновения конфликта правил основная таблица стилей (из файла, в который осуществляется импорт) имеет приоритет над импортируемыми таблицами стилей. Если же вы импортируете несколько таблиц стилей, правила из таблицы стилей, импортированной последней, имеют приоритет над правилами из ранее импортированных таблиц стилей.

7.2.7. Задание значений URL

URL представляет собой стандартный Internet-адрес, такой как <http://mspress.microsoft.com/>. Директива @import и свойство background-image требуют указания значения URL для задания местонахождения соответствующего ресурса (таблицы стилей или файла изображения). URL задается с использованием следующей формы записи, где URL есть URL. Обратите внимание, что нельзя помещать пробелы между url и символом открывающей скобки (url(URL)).

Вы можете использовать полностью заданный URL, как в следующих примерах:

```
@import url(http://www.my_domain.com/stylesheets/MyStyles.css);
INVENTORY
{background-image:url(file:///E:\Example
Code\Background.gif) }
```

Или вы можете использовать частичный URL, который задает местонахождение относительно местонахождения файла таблицы стилей, содержащего URL. Относительные URL в таблицах стилей работают подобно URL в HTML-страницах. Например, если файл таблицы стилей находится в папке Example Code, следующий относительный URL будет эквивалентен полному URL из предыдущего примера (а именно, file:///E:\Example Code\Background.gif):

```
INVENTORY {background-image:url (Background.gif) }
```

Чтобы связать таблицу каскадных стилей с XML-документом, вы должны вставить в документ зарезервированную инструкцию по обработке xml-stylesheet. Эта инструкция по обработке имеет следующую обобщенную форму записи, где CSSFilePath есть адрес, задающий местонахождение файла таблицы стилей:

```
<?xml-stylesheet type="text/css" href=CSSFilePath?>
```

Вы можете использовать полный URL, например:

```
<?xml-stylesheet type="text/css"
    style="color:#666666;"
href="/external/?popup=0&url=http%3A%2F%2Fwww.my_domain.
                                com%2FInventory01.css"
onMouseOver="menuSetHelpText ('external');
return false;" onMouseOut="menuClearHelpText ()" ?>
```

Чаще применяется частичный URL, который задает местонахождение относительно местонахождения XML-документа, содержащего инструкцию по обработке xml-stylesheet, например:

```
<?xml-stylesheet type="text/css" href="Inventory01.css"?>
```

Обычно вы добавляете инструкцию по обработке xml-stylesheet в пролог XML-документа, вслед за объявлением XML, как вы видели в примере XML-документа в листинге 7.2 (см. на с. 127–128).

Возможность присоединять к XML-документу внешнюю таблицу стилей увеличивает гибкость форматирования документа. Вы можете полностью изменить вид документа, просто присоединив к нему другую таблицу стилей. Чтобы сделать это, достаточно всего лишь отредактировать URL в инструкции по обработке xml-stylesheet, не внося никаких других изменений в XML-документ.

Если вы связали таблицу стилей с XML-документом, вы можете открыть этот документ непосредственно в Internet Explorer 5. На-

пример, вы можете ввести URL документа или путь к файлу в поле Address (Адрес), как показано на рис. 7.5, и нажать клавишу <Enter>.

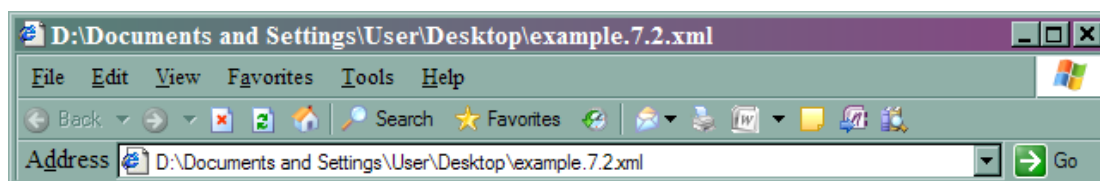


Рис. 7.5. Вывод пути к файлу в поле Address

Или, предполагая, что Internet Explorer 5 есть ваш браузер, используемый по умолчанию, вы можете просто дважды щелкнуть на имени файла XML-документа в окне Windows Explorer (Проводник) или в окне папки.

⇒ *Примечание.* Если браузер не может найти файл таблицы стилей, заданный в инструкции по обработке `xml-stylesheet`, он отобразит текст документа с использованием своего собственного набора свойств (например, с текущими значениями гарнитуры и размера шрифта). Если XML-документ не связан с таблицей стилей (т. е. документ не содержит инструкции по обработке `xml-stylesheet`), то Internet Explorer 5 отобразит исходный XML-код для документа, но не содержимое документа.

Вы можете включить в XML-документ более одной таблицы стилей, вставив для каждой из них инструкцию по обработке `xmlstylesheet` в начале XML-документа, например:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="Book01.css"?>
<?xml-stylesheet type="text/css" href="Book02.css"?>
<INVENTORY>
    <!-- Содержимое элемента Документ. -->
</INVENTORY>
```

Возможность связывания с несколькими таблицами стилей позволяет вам хранить группы взаимосвязанных правил в отдельных файлах, а затем объединять их при создании документов определенных типов.

Если вы устанавливаете связи с несколькими таблицами стилей, Internet Explorer 5 объединяет правила из различных таблиц. Если отдельные таблицы стилей содержат конфликтующие правила,

правила из последней связываемой с документом таблицы стилей имеют приоритет над правилами, содержащимися в предшествующих таблицах стилей.

7.2.8. Присвоение значений в таблицах каскадных стилей

В таблицах каскадных стилей вы можете присваивать значения свойствам на нескольких различных уровнях (подобно каскаду водопада, в котором поток падает, проходя по нескольким ступеням). Ниже дано описание основных уровней, на которых вы можете присваивать значение свойству. Уровни представлены в порядке их приоритетов – от высшего к низшему. Когда браузер готовится отобразить элемент, и значение его определенного свойства, например `font-size`, конфликтует со значением, присвоенным этому элементу на других уровнях, браузер использует значение, присвоенное на наивысшем уровне приоритета.

Если вы присвоили значение свойству в атрибуте `STYLE` для определенного элемента в XML-документе, браузер применяет это значение при отображении элемента. Например, он отобразит следующий элемент полужирным:

```
<TITLE STYLE="font-weight:bold">Leaves of Grass</TITLE>
```

Если вы не установили свойство в атрибуте `STYLE`, браузер использует значение свойства, объявленного в правиле CSS с контекстуальным селектором (т. е. селектором, который определяет элемент с одним или несколькими его элементами-предками). Предположим, что следующий элемент является элементом Документ XML-документа:

```
<MAPS>
  <CITY>
    <NAME>Santa Fe</NAME>
    <STATE>New Mexico</STATE>
  </CITY>
  <STATE>California</STATE>
</MAPS>
```

Допустим также, что присоединенная таблица стилей содержит следующие правила:

```
CITY STATE
  {font-style:normal}
STATE
  {font-style:italic}
```

Браузер использует правило CITY STATE для форматирования элемента «New Mexico» STATE, поскольку оно имеет контекстуальный селектор и, следовательно, характеризуется приоритетом над правилом STATE, имеющим родовой селектор. Надпись «New Mexico» в результате будет отображена обычным шрифтом.

Если вы не объявили значение определенного свойства в правиле, имеющем соответствующий контекстуальный селектор, браузер использует значение, объявленное в правиле с родовым селектором (т. е. селектором, который включает только имя элемента). Например, для второго компонента рассматриваемой таблицы стилей браузер не найдет соответствующего контекстуального правила для элемента «California» STATE, поэтому использует родовое правило STATE, в результате чего надпись «California» будет отображена курсивом.

Если вы не объявили значение для определенного свойства для элемента в родовом правиле, браузер использует установку свойства, объявленную для ближайшего элемента-предка (родителя, родителя родителя и т. д.). Например, в таблице стилей из листинга 7.1 правило для элемента TITLE не присваивает значение для свойства font-size:

```
TITLE
    {font-style:italic}
```

Следовательно, браузер будет использовать установку свойства font-size для родительского элемента BOOK (элемент BOOK является родителем для элемента TITLE в XML-документе, использующем таблицу стилей):

```
BOOK
    {display:block;
    margin-top:12pt;
    font-size:10pt}
```

В результате текст элемента TITLE будет отображен с размером шрифта 10 пт.

Этот процесс имеет место только для наследуемого свойства. Для ненаследуемого свойства браузер будет использовать значение свойства по умолчанию.

Если таблица стилей не содержит установку свойства для какого-либо родительского элемента, браузер использует свою собственную установку. Такой установкой может быть значение

по умолчанию, встроенное в браузер, либо значение, заданное пользователем браузера.

Например, поскольку в рассматриваемой таблице стилей из листинга 7.1 (см. на с. 127) не установлено значение для свойства `font-family`, браузер использует свое собственное значение этого свойства для отображения всех элементов. (В Internet Explorer 5 это шрифт Times New Roman, если только пользователь браузера не выберет другое семейство шрифтов, воспользовавшись командой Internet Options (Свойства обозревателя) из меню Tools (Сервис).)

Этот процесс имеет место только для наследуемых свойств. Для ненаследуемых свойств браузер использует значение свойства по умолчанию. Как вы можете видеть из этого списка, основной принцип здесь следующий: если вы присвоили свойству значения, конфликтующие на различных уровнях, браузер отдает предпочтение более конкретизированному правилу. Например, установленное для элемента свойство является само по себе более конкретизированным, чем установка для родителя элемента, поэтому имеет приоритет. Вы можете воспользоваться этим принципом и в более сложных случаях.



Примечание. Порядок приоритетов не является незыблемым. Можно сделать так, что установка свойства в браузере будет иметь приоритет над установкой свойства в таблице стилей, присоединенной к вашему XML-документу. Это дает возможность пользователям управлять форматированием (например, пользователь с ослабленным зрением может использовать увеличенный шрифт). В Internet Explorer 5, например, пользователь может присвоить установке свойств в браузере наивысший приоритет по отношению к установкам свойств в таблице стилей, выбрав команду Options (Свойства обозревателя) из меню Tools (Сервис), щелкнув на кнопке Accessibility (Оформление) на вкладке General (Общие) в диалоговом окне Internet Options (Свойства обозревателя) и выбрав соответствующие опции.

Что произойдет, если для определенного свойства установлены конфликтующие значения на одном и том же уровне? В таком случае браузер использует последнюю установку, которую он обработал. Например, если два родовых правила для одного элемента имеют конфликтующие установки для свойства `font-style`, как в следующем примере, браузер использует второе из них, поскольку оно обрабатывается последним:

```
TITLE, AUTHOR, BINDING, PRICE
    {display:block;
    font-size:12pt;
    font-weight:bold;
    font-style:italic}
AUTHOR
    {font-style:normal}
```


В этом примере элементы AUTHOR будут отформатированы обычным шрифтом, а не курсивом.

Ниже приведен порядок, в котором браузер обрабатывает правила таблицы стилей:

- если вы связываете несколько таблиц стилей с документом, использующим инструкцию по обработке xml-styleSheet, браузер обрабатывает таблицы стилей в последовательности, в которой они приведены в инструкции по обработке;

- если вы импортируете одну или несколько таблиц стилей в другую таблицу стилей с использованием директивы @import, браузер обрабатывает импортированные таблицы стилей перед таблицей, в которую они импортируются. Порядок обработки при этом определяется порядком импорта;

- в таблице стилей правила обрабатываются в том порядке, в котором они записаны.

 *Примечание.* Правило, согласно которому браузер использует последнюю установку свойства, противоположно правилу, применяемому XML-процессором в случае наличия множественных объявлений атрибутов или примитивов. XML-процессор использует первое объявление атрибута или примитива и игнорирует все последующие.

7.2.9. Установка свойства display


Свойство display управляет основным способом отображения текста элемента браузером. Вы можете назначить ему три ключевых слова CSS:

- 1) block. Браузер всегда помещает пустую строку перед и после текста элемента (который включает и текст, принадлежащий любым дочерним элементам). В результате текст элемента отображается в отдельном «блоке» с предшествующим текстом выше и последующим текстом ниже. Присвоение значения block позволяет вам форматировать текст с применением различных

свойств оформления к блоку текста, таких как поля, рамки и отступы. Элемент `block` похож на абзац в программе текстового процессора, который отделен пробелами от предшествующего и последующего текста и для которого можно задавать отступы, рамки и т. д.;

2) `inline` (по умолчанию). Браузер не вставляет пустую строку перед или после текста элемента. Он будет вставлять пустые строки внутри текста элемента только с целью уместить текст в окне. Текст элемента, таким образом, может быть размещен в той же строке, что и предыдущий или последующий текст. Элемент `inline` аналогичен группе символов внутри абзаца в программе текстового процессора;

3) `none`. Браузер не отображает элемент. Вы можете использовать эту установку для элементов, несущих информацию, которую вы не хотели бы помещать на экране.

 *Примечание.* В спецификации CSS указано, что свойство `display` не наследуется дочерними элементами. Это так, если вы назначаете установку `block` для свойства `display` элемента. Однако элементы вполне эффективно наследуют установку `none`, поскольку, когда вы назначаете эту установку свойству `display` элемента-родителя, вы тем самым скрываете и дочерние элементы. Дочерние элементы элемента `inline` также будут элементами `inline`, если для них не установлено свойство `display`, так как `inline` является значением по умолчанию.

Информация относительно назначения ключевых слов CSS свойствам приведена далее в 7.2.10 «Задание ключевых слов CSS в качестве значений».

Предположим, вы используете следующую таблицу стилей для отображения XML-документа, представленного в листинге 7.2 (напомним, что для изменения таблицы стилей, применяемой для отображения XML-документа, вам следует отредактировать инструкцию по обработке `xml-styleSheet` в документе):

```
BOOK
{display:block;
margin-top:12pt;
font-size:10pt}
TITLE
{font-style:italic}
```



```
AUTHOR
    {font-weight:bold}
PAGES
    {display:none}
```

Поскольку свойству `display` элемента `BOOK` присвоено значение `block`, браузер всегда будет помещать пустую строку перед и после текста элемента.

В силу того, что таблица стилей не присваивает значения свойству `display` для элементов `TITLE`, `AUTHOR`, `BINDING` и `PRICE` (и эти элементы не наследуют значение свойства `display` от их родительских элементов), браузер воспримет их как элементы `inline`, что является установкой по умолчанию. Следовательно, браузер не будет помещать пустые строки между этими элементами и, если допустить, что окно браузера имеет достаточную ширину, отобразит их все на одной строке. На рис. 7.6 показано, как должен выглядеть результат.

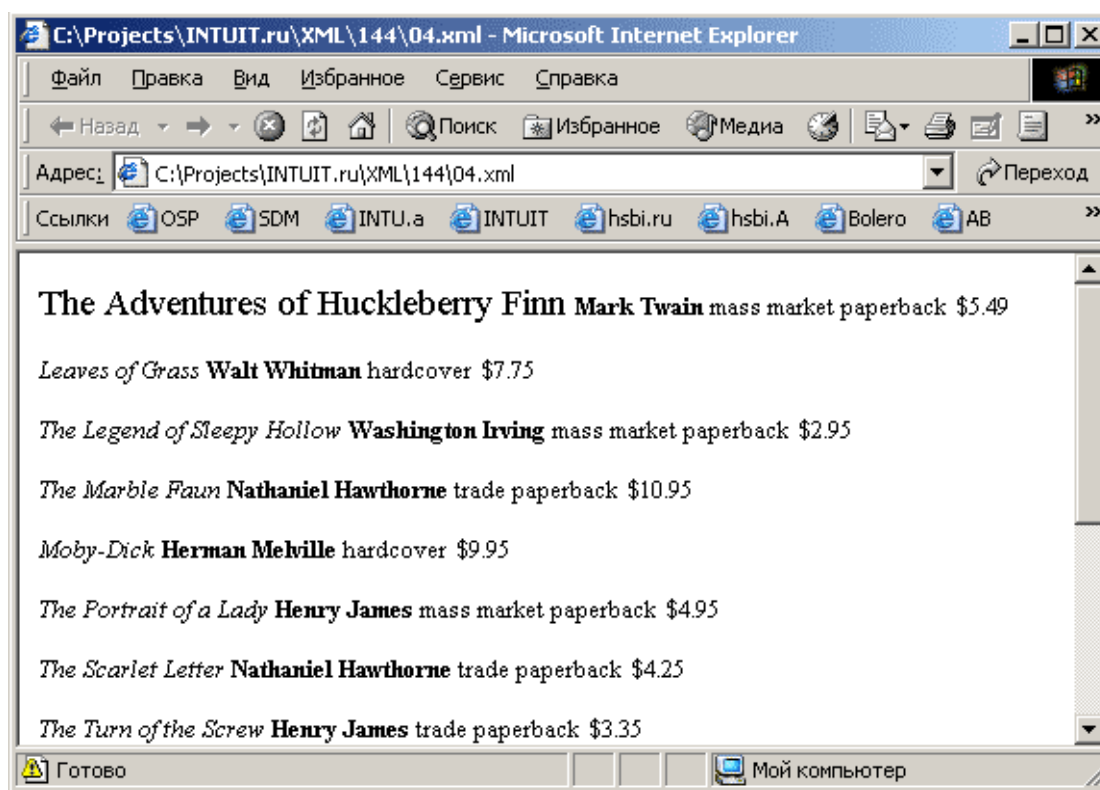


Рис. 7.6. Отображение в Internet Explorer 5 страницы, для которой установлены элементы по умолчанию

Поскольку для свойства `display` элемента `PAGES` установлено значение `none`, браузер не отобразит этот элемент.

7.2.10. Задание ключевых слов CSS в качестве значений

Для многих свойств CSS вы можете или должны присваивать значение с использованием predetermined ключевых слов CSS. Специфические ключевые слова, которые вы можете использовать, определяются особенностью свойства. Например, вы можете назначить свойству `display` одно из трех ключевых слов: `block`, `inline` или `none`. Свойству `color` вы можете назначить одно из 16 ключевых слов, которые описывают основные цвета, такие как `red` (красный), `green` (зеленый), `yellow` (желтый) или `fuchsia` (фуксия), как в следующем примере:

```
PARA {color:fuchsia}
```

Свойству `border-style` вы можете назначить одно из 9 ключевых слов: `solid`, `dotted`, `dashed`, `double`, `groove`, `ridge`, `inset`, `outset` или `none`. Например:

```
SECTION {border-style:solid}
```

7.3. УСТАНОВКА СВОЙСТВ ШРИФТА

В стандартной CSS-таблице предусмотрены следующие свойства, определяющие вид шрифта, который используется для отображения текста элемента:

- `font-family`;
- `font-size`;
- `font-style`;
- `font-weight`;
- `font-variant`.

Все эти свойства наследуются дочерними элементами.

7.3.1. Установка свойства `font-family`

Свойство `font-family` определяет имя шрифта (гарнитуру), используемого для отображения текста элемента. Например:

```
BOOK {font-family:Arial}
```

Вы можете задавать любое имя шрифта по вашему желанию. Если браузер не может найти требуемый шрифт, он заменит его на другой доступный шрифт.



Совет. Если имя шрифта содержит пробелы, заключите название в кавычки, как в следующем примере:

```
BOOK {font-family:"Times New Roman"}
```

Вы можете расширить возможность выбора и привести несколько вариантов допустимых к использованию шрифтов в порядке приоритета, разделяя их запятыми. Например:

```
BOOK {font-family:Arial, Helvetica}
```

Если шрифт Arial не доступен, браузер использует шрифт Helvetica. Если шрифт Helvetica также не доступен, он заменит его на какой-либо другой имеющийся шрифт.

Вы можете еще больше расширить возможность выбора нужного шрифта, включив в описание ключевое слово CSS (обычно в конце списка), указывающее на общий тип шрифта, который вы хотите использовать. Например:

```
BOOK {font-family:Arial, Helvetica, sans-serif}
```

В этом случае, если браузер не найдет шрифтов Arial или Helvetica, он использует какой-либо другой шрифт без засечек (sans-serif).

В табл. 7.1 приведен перечень ключевых слов, которые вы можете использовать для указания общего типа нужного вам шрифта. В спецификации CSS они носят название имен родовых семейств. Для каждого имени родового семейства в таблице также представлено имя определенного шрифта, принадлежащего к этому семейству, а также образцы отображения текста для указанного семейства.

Таблица 7.1

Перечень ключевых слов, используемых для указания нужного типа шрифта

Ключевое слово имени родового семейства для свойства font-family	Пример соответствующего шрифта
serif	Times New Roman
sans-serif	Arial
cursive	<i>ZapfChancery</i>
fantasy	Western
monospace	Courier New

Например, если вы присоединили следующую таблицу стилей к XML-документу из листинга 7.2, Internet Explorer 5 отобразит документ, как показано на рис. 7.7:

```
BOOK
    {display:block;
    margin-top:12pt;
    font-family:Arial, sans-serif;
    font-size:12pt}
TITLE
    {font-style:italic}
AUTHOR
    {font-family:"Times New Roman", serif}
```

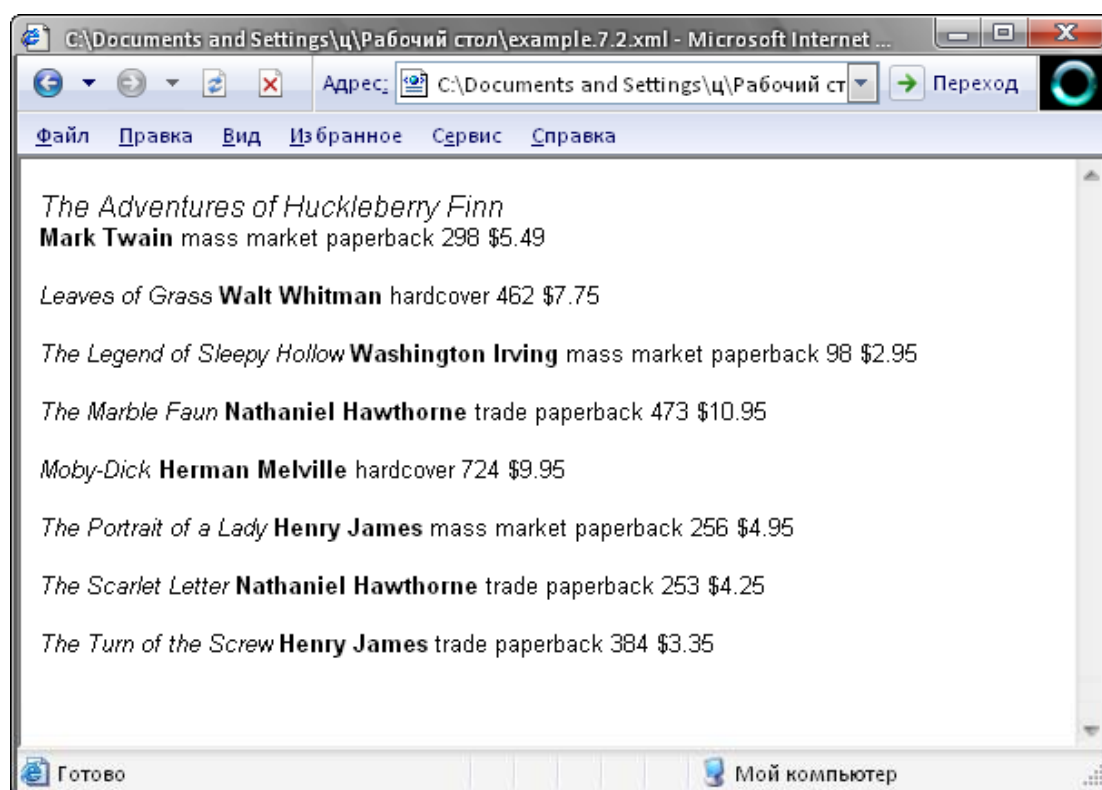


Рис. 7.7. Отображение в Internet Explorer 5 XML-документа в соответствии с листингом 7.2

Шрифт Arial, назначенный свойству font-family элемента BOOK, наследуется всеми дочерними элементами, за исключением элемента AUTHOR, для которого задано свое значение свойства font-family.

7.3.2. Установка свойства font-size

Свойство font-size устанавливает высоту шрифта, используемого для отображения текста элемента. Вы можете присваивать этому свойству четыре различных типа значений.

1. Значение, задающее размер относительно размера шрифта браузера. Вы можете задать размер шрифта относительно текущего размера шрифта браузера, присвоив свойству font-size одно из значений в виде ключевых слов, представленных в табл. 7.2. Для Internet Explorer 5 значение small заставит браузер использовать текущий выбранный размер шрифта; другие значения масштабируются с увеличением или уменьшением относительно этого размера.

Таблица 7.2

Ключевые слова для свойства font-size, задающего размер относительно размера шрифта браузера

Ключевое слово для font-size	Пример правила CSS	Описание
xx-small	TITLE {font-size:xx-small}	Наименьший размер шрифта, который может задаваться с помощью ключевого слова
x-small	TITLE {font-size:x-small}	Приблизительно в 1,5 раза больше, чем xx-small
small	TITLE {font-size:small}	Приблизительно в 1,5 раза больше, чем x-small. Это значение предписывает Internet Explorer 5 использовать его текущий размер шрифта
medium	TITLE {font-size:medium}	Приблизительно в 1,5 раза больше, чем small
large	TITLE {font-size:large}	Приблизительно в 1,5 раза больше, чем medium
x-large	TITLE {font-size:x-large}	Приблизительно в 1,5 раза больше, чем large
xx-large	TITLE {font-size:xx-large}	Приблизительно в 1,5 раза больше, чем x-large

⇒ *Примечание.* Спецификация CSS рекомендует использовать масштабный коэффициент 1,5. Однако в Internet Explorer 5 реальное соотношение между различными значениями размера отличается в меньшей степени. Например, medium в действительности составляет примерно 1,15 от small.

2. Значение, задающее размер относительно размера родительского шрифта. Вы можете задать размер шрифта относительно текущего размера шрифта для родительского элемента, присвоив свойству font-size одно из следующих значений с помощью ключевых слов из табл. 7.3.

Таблица 7.3

**Ключевые слова для свойства font-size, задающего размер
относительно размера родительского шрифта**

Ключевое слово для font-size	Пример правила CSS	Описание
smaller	TITLE {font-size:smaller}	Приблизительно на 33% меньше размера шрифта для родительского элемента (или для корневого элемента на 33% меньше размера шрифта браузера)
larger	TITLE {font-size:larger}	Приблизительно на 50% больше размера шрифта для родительского элемента (или для корневого элемента на 50% больше размера шрифта браузера)

⇒ *Примечание.* Значения 33 и 50%, приведенные в табл. 7.3, основаны на масштабном коэффициенте 1,5, рекомендованном спецификацией CSS. В действительности результат может оказаться иным.

3. Задание размера в процентах от размера родительского шрифта. Вместо того чтобы использовать ключевые слова smaller или larger, вы можете задать размер шрифта в процентах относительно текущего размера шрифта родительского элемента с большей степенью точности, задав свойству font-size значение в процентах. Например, следующее правило устанавливает размер шрифта в 1,5 раза больше, чем размер шрифта родительского элемента:

```
TITLE {font-size:150%}
```

(Если браузер использует рекомендуемый масштабный коэффициент 1,5, это правило будет эквивалентно правилу TITLE {font-size:larger}.)

Следующее правило устанавливает размер шрифта немного больше, чем установленный предыдущим правилом:

```
TITLE {font-size:160%}
```

Заметим, что для корневого элемента проценты берутся относительно размера шрифта браузера.

4. Задание численных значений размера. Вы также можете задать размер шрифта для элемента, присвоив свойству font-size абсолютное значение. Например, следующее правило устанавливает размер шрифта в 12 пт:

```
TITLE {font-size:12pt}
```

А это правило устанавливает размер шрифта в 2 раза больше, чем размер шрифта родительского элемента:

```
TITLE {font-size:2em}
```

(Этот пример эквивалентен записи `TITLE {font-size:200%;}`.)

7.3.3. Задание значений в процентах

Вы можете задавать для определенных свойств значения в процентах. Это необходимо, если вы имеете дело с относительными размерами, а не с действительными абсолютными размерами.

Значение в процентах задает размер для некоторого свойства как процент от какого-либо значения. Какое это будет значение, зависит от того свойства, для которого выполняется назначение. Обычно это размер шрифта самого элемента. Например, следующее назначение устанавливает высоту строки в 2 раза большей, чем высота текущего шрифта элемента, в результате получаются строки с двойным интервалом:

```
SECTION {line-height:200%}
```

Для свойства `font-size`, однако, значение в процентах берется относительно текущего размера шрифта родительского элемента. Например, следующее правило устанавливает для элемента высоту шрифта, составляющую три четверти от высоты текущего шрифта его родительского элемента:

```
PARAGRAPH {font-size:75%}
```

Заметим, что если элемент наследует значение свойства в процентах, наследуется вычисленный результат, но не сама величина в процентах. (Другими словами, если ряд нисходящих элементов наследует значение в процентах, то размер для каждого последующего уровня будет уменьшаться или увеличиваться.)

7.3.4. Задание значений в размерных единицах

Для многих свойств CSS могут или должны быть присвоены значения, выраженные в единицах размеров. Такие значения задают размеры шрифтов, положения фоновых изображений, расстояния между символами, отступы, междустрочные интервалы,

величины полей и рамок, высоту и ширину элементов и другие свойства. Вы можете присвоить абсолютное или относительное численное значение любому свойству, принимающему численное значение.

Абсолютное значение размера определяет точный размер с использованием стандартных единиц измерения, таких как дюймы, пункты или миллиметры. В табл. 7.4 приведен список различных единиц измерения, которыми вы можете пользоваться. Для каждой единицы представлена аббревиатура, которую вы должны применить для указания единиц измерения в правиле.

Таблица 7.4

Абсолютные размерные единицы

Аббревиатура	Сантиметры	Дюймы	Миллиметры	Пики	Пункты
cm	1,0	0,3937	10,0	2,3622	28,3465
in	2,54	1,0	25,4	6,0	72,0
mm	0,1	0,03937	1,0	0,23622	2,83465
pc	0,42333	0,16667	4,23333	1,0	12,0
pt	0,03528	0,01389	0,35278	0,08333	1,0

Например, следующие два правила присваивают абсолютные значения размеров:

```
STANZA {font-size:12pt}
PARAGRAPH {margin-top:.25in}
```

Относительное значение задает размер относительно высоты текущего шрифта элемента или относительно размера пикселя на экране монитора, который используется для отображения документа. (Пиксель – одна из отдельных точек, составляющих изображение на мониторе компьютера или экране телевизора.) В табл. 7.5 представлены различные виды относительных единиц размера, которые вы можете использовать.

Таблица 7.5

Относительные размерные единицы

Единица	Размерность единицы
em	Высота текущего шрифта элемента
ex	Высота строчной буквы x текущего шрифта элемента
px	Размер пикселя на мониторе

Имеется одно исключение: когда вы присваиваете свойству `font-size` значение в единицах `em` или `ex`, оно берется относительно размеров шрифта родительского элемента.

Например, следующее правило добавляет к элементу отступ сверху. Высота поля отступа будет равна высоте шрифта элемента:

```
BOOK {margin-top:1em}
```

Нижеприведенное правило создает верхнее поле высотой 15 пикселей:

```
SECTION {margin-top:15px}
```

А это правило устанавливает высоту шрифта элемента в три четверти от высоты шрифта его родительского элемента:

```
PARAGRAPH {font-size:.75em}
```

Заметим, что если дочерний элемент наследует относительное значение, наследуется результат вычисления, а не само относительное значение.

7.3.5. Установка свойства `font-style`

Свойство `font-style` управляет стилем начертания символов — обычным или курсивным. Вы можете присвоить этому свойству одно из следующих значений в виде ключевых слов, приведенных в табл. 7.6.

Таблица 7.6

Ключевые слова для свойства `font-style`

Ключевое слово для <code>font-style</code>	Пример правила CSS	Описание
<code>italic</code>	<code>TITLE {font-style:italic}</code>	Назначает курсивное начертание для шрифта, если это возможно, в противном случае назначает наклонное начертание
<code>oblique</code>	<code>TITLE {font-style:oblique}</code>	Назначает наклонное начертание шрифта, если это возможно (Данное начертание образуется путем наклона символов обычного шрифта.)
<code>normal</code>	<code>TITLE {font-style:normal}</code>	Назначает обычный (романский) шрифт

7.3.6. Установка свойства font-weight

Свойство font-weight определяет, насколько плотными (т. е. насколько темными и жирными) будут символы элемента. Вы можете присвоить этому свойству одно из следующих значений в виде ключевых слов, представленных в табл. 7.7.

Таблица 7.7

Ключевые слова для свойства font-weight

Ключевое слово для font-weight	Пример правила CSS	Описание
normal	TITLE {font-weight:normal}	Отображает текст с нормальной степенью плотности
bold	TITLE {font-weight:bold}	Отображает символы стандартным полужирным начертанием
bolder	TITLE {font-weight:bolder}	Отображает текст более жирным шрифтом, чем шрифт родительского элемента (или для корневого элемента – чем шрифт браузера)
100	TITLE {font-weight:100}	Отображает текст с самой малой плотностью, доступной для данного шрифта
200	TITLE {font-weight:200}	Последующие значения (200–900) отображают текст с возрастающей степенью плотности
300	TITLE {font-weight:300}	
400	TITLE {font-weight:400}	
500	TITLE {font-weight:500}	
600	TITLE {font-weight:600}	
700	TITLE {font-weight:700}	
800	TITLE {font-weight:800}	
900	TITLE {font-weight:900}	

Возможно, браузер будет не способен отобразить все эти степени плотности.

7.3.7. Установка свойства font-variant

Вы можете использовать свойство font-variant для преобразования всех символов текста в прописные буквы. Назначьте этому свойству одно из ключевых слов, указанных в табл. 7.8.

Ключевые слова для свойства font-variant

Ключевое слово для font-variant	Пример правила CSS	Описание
small-caps	TITLE {font-variant:small-caps}	Преобразует символы текста в прописные
normal	TITLE {font-variant:normal}	Оставляет оригинальный вид текста с сочетанием строчных и прописных букв (т. е. текст не преобразуется)

7.4. УСТАНОВКА СВОЙСТВА COLOR

Свойство color устанавливает цвет текста элемента. Вы можете присваивать этому свойству значение цвета с использованием форматов, которые описаны в пункте «Задание значений цвета» далее в этой главе. Например, следующее правило устанавливает синий цвет для текста элемента AUTHOR:

```
AUTHOR {color:blue}
```

А это правило устанавливает для текста элемента AUTHOR красный цвет:

```
AUTHOR {color:rgb(255,0,0)}
```

Свойство color наследуется дочерними элементами. Так, если вы присоедините следующую таблицу стилей к XML-документу из листинга 7.2, весь текст будет отображен синим цветом за исключением текста элемента PRICE, который будет отображен красным, поскольку для него в таблице стилей предусмотрена отдельная установка цвета:

```
BOOK
{display:block;
margin-top:12pt;
font-size:10pt;
color:blue}
TITLE
{font-style:italic}
AUTHOR
```

```
{font-weight:bold}  
PRICE  
{color:red}
```



Совет. Свойство `color` устанавливает цвет для отдельных букв текста (foreground color). Чтобы установить цвет фона, воспользуйтесь свойством `background-color`.

Задание значений цвета

К свойствам, которым вы можете назначать значения цвета, относятся `color`, `background-color` и `border-color`. Вы можете присвоить значение цвета с использованием четырех различных форматов, которые содержатся в приведенных ниже примерах правил. Эти правила являются эквивалентными, т. е. каждое из них назначает свойству `color` красный цвет:

```
PARA {color:red}  
PARA {color:rgb(255,0,0)}  
PARA {color:#FF0000}  
PARA {color:rgb(100%,0%,0%)}
```

Первый формат использует ключевое слово CSS (`red`), в то время как другие три формата задают цвет путем установки относительной интенсивности компонентов красного, зеленого и синего в составе цвета. Во втором формате интенсивность каждого из цветов задается десятичным числом в диапазоне от 0 до 255. В третьем формате цвет задается с использованием шестизначного шестнадцатеричного числа в диапазоне от 000000 до FFFFFFFF, где первые две цифры определяют интенсивность красного, вторые две цифры — интенсивность зеленого, а последние две цифры — интенсивность синего. В последнем формате интенсивность каждого из цветов задается в процентах от 0 до 100%.

В табл. 7.9 приведены значения цвета, которые вы можете присвоить с помощью ключевых слов CSS. Для каждого цвета указано описание.

Если вы используете один из RGB-форматов, вы можете создать и множество других цветов, не показанных в табл. 7.9. Фактически вы можете присвоить каждому из компонентов 256 различных значений, что в сумме дает 16 777 216 различных цветов ($256 \times 256 \times 256$).

Задание значений цвета

Цвет	Ключевое слово CSS	Десятичный RGB-формат	Шестнадцатеричный RGB-формат	Процентный RGB-формат
Красный	red	rgb(255,0,0)	#FF0000	rgb(100%,0%,0%)
Коричневый	maroon	rgb(128,0,0)	#800000	rgb(50%,0%,0%)
Светло-зеленый	lime	rgb(0,255,0)	#00FF00	rgb(0%,100%,0%)
Зеленый	green	rgb(0,128,0)	#008000	rgb(0%,50%,0%)
Голубой	blue	rgb(0,0,255)	#0000FF	rgb(0%,0%,100%)
Темно-синий	navy	rgb(0,0,128)	#000080	rgb(0%,0%,50%)
Светло-желтый	yellow	rgb(255,255,0)	#FFFF00	rgb(100%,100%,0%)
Темно-желтый	olive	rgb(128,128,0)	#808000	rgb(50%,50%,0%)
Светло-голубой	aqua	rgb(0,255,255)	#00FFFF	rgb(0%,100%,100%)
Темно-голубой	teal	rgb(0,128,128)	#008080	rgb(0%,50%,50%)
Светло-фиолетовый	fuchsia	rgb(255,0,255)	#FF00FF	rgb(100%,0%,100%)
Темно-фиолетовый	purple	rgb(128,0,128)	#800080	rgb(50%,0%,50%)
Белый	white	rgb(255,255,255)	#FFFFFF	rgb(100%,100%,100%)
Черный	black	rgb(0,0,0)	#000000	rgb(0%,0%,0%)
Светло-серый	silver	rgb(192,192,192)	#C0C0C0	rgb(75%,75%,75%)
Темно-серый	gray	rgb(128,128,128)	#808080	rgb(50%,50%,50%)

7.5. УСТАНОВКА СВОЙСТВ ФОНА

Стандарт CSS поддерживает следующие свойства, позволяющие вам модифицировать фоновое оформление элемента:

- background-color;
- background-image;
- background-repeat;
- background-position.

Фон представляет собой область, окружающую отдельные символы текста элемента. В качестве фона вы можете устанавливать либо сплошной цвет, либо рисунок.

Технически дочерние элементы не наследуют свойств фона. Однако по умолчанию фон элемента является прозрачным. Это означает, что если вы опустите все свойства фона для дочернего элемента, будут видны цвет фона или рисунок родительского элемента.

7.5.1. Установка свойства background-color

Вы можете установить цветовой фон для элемента (заливку), присвоив значение цвета свойству background-color. Например, следующее правило устанавливает светло-желтый цвет фона для элемента TITLE:

```
TITLE {background-color:yellow}
```

Напомним, что свойство color устанавливает цвет собственно символов элемента. Так, следующее правило создает синие буквы на желтом фоне:

```
TITLE
  {color:blue;
  background-color:yellow}
```

Если вы не хотите задавать сплошной цвет фона для элемента, вы можете присвоить свойству background-color значение transparent, например:

```
TITLE {background-color:transparent}
```

7.5.2. Установка свойства background-image

Вы можете добавить элементу фоновый рисунок, назначив свойству background-image URL файла с рисунком. Например, следующее правило устанавливает для элемента STANZA фоновый рисунок, содержащийся в файле Leaf.bmp:

```
STANZA {background-image:url(Leaf.bmp)}
```

Для дальнейшего рассмотрения воспользуемся таблицей стилей, которая представлена в листинге 7.3 и присоединена к XML-документу, приведенному в листинге 7.4.

```
Листинг 7.3. Leaves.css
/* File Name: Leaves.css */
POEM
  {font-size:145%}
POEM, TITLE, SUBTITLE, AUTHOR, SECTION, NUMBER, STANZA, VERSE
  {display:block}
SECTION, STANZA
  {margin-top:1em}
STANZA
  {background-image:url(Leaf.bmp)}
```

Листинг 7.4. Leaves.xml

```
<?xml version="1.0"?>
<!-- File Name: Leaves.xml -->
<?xml-stylesheet type="text/css" href="Leaves.css"?>
<POEM>
  <TITLE>Leaves of Grass
  <SUBTITLE>I Sing the Body Electric</SUBTITLE>
</TITLE>
  <AUTHOR>by Walt Whitman</AUTHOR>
  <SECTION>
    <NUMBER>1.</NUMBER>
  <STANZA>
    <VERSE>I SING the Body electric;</VERSE>
    <VERSE>The armies of those I love engirth me, and
I engirth them; </VERSE>
    <VERSE>They will not let me off till I go with
them, respond to them,</VERSE>
    <VERSE>And discorrupt them, and charge them full
with the charge of the Soul.</VERSE>
  </STANZA>
  <STANZA>
    <VERSE>Was it doubted that those who corrupt their
own bodies conceal themselves;</VERSE>
    <VERSE>And if those who defile the living are as
bad as they who defile the dead?</VERSE>
    <VERSE>And if the body does not do as much as the
Soul?</VERSE>
    <VERSE>And if the body were not the Soul, what is
the Soul?</VERSE>
  </STANZA>
</SECTION>
</POEM>
```

На рис. 7.8 представлено содержимое графического файла Leaf.bmp.



Рис. 7.8. Содержимое графического файла Leaf.bmp

Internet Explorer 5 отобразит документ Leaves.xml, как показано на рис. 7.9.

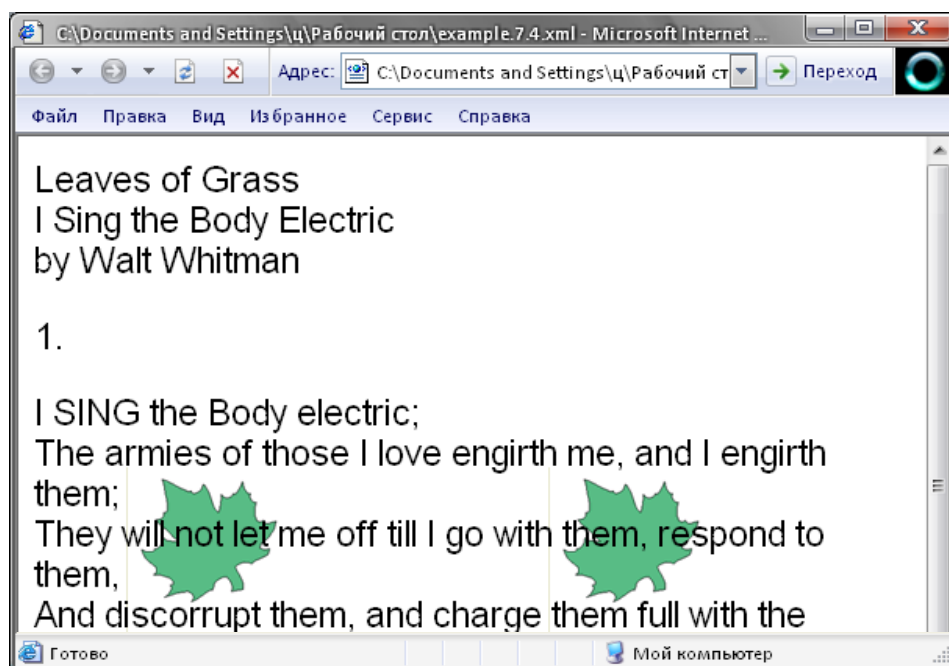


Рис. 7.9. Отображение в Internet Explorer 5 файла Leaves.xml

Обратите внимание, что рисунок повторяется при заполнении всей области, занимаемой содержимым элемента, достигая почти правой границы окна браузера. Любая часть изображения, выступающая за пределы текста элемента, обрезается. В рассматриваемом примере обрезается только очень небольшая часть изображений в нижнем ряду каждого элемента STANZA.

Если вы не хотите задавать фоновый рисунок для элемента, то можете установить для свойства `background-image` значение `none`, например:

```
STANZA {background-image:none}
```

Либо, поскольку `none` является значением по умолчанию, вы можете опустить свойство `background-image` для элемента. Если вы не назначили для элемента сплошную заливку (без рисунка), установка `none` приведет к тому, что будет виден фон родительского элемента (или браузера).

7.5.3. Установка свойства `background-repeat`

Если вы назначили свойству `background-image` файл рисунка, вы можете управлять повторами изображения, назначив свойству `background-repeat` одно из следующих ключевых слов:

1) repeat (по умолчанию). Повторяет изображения как по горизонтали, так и по вертикали. Добавление background-repeat к правилу STANZA в таблице стилей из листинга 7.3 не окажет эффекта на способ отображения документа (рис. 7.10):

STANZA

```
{background-image:url(Leaf.bmp);  
background-repeat:repeat}
```

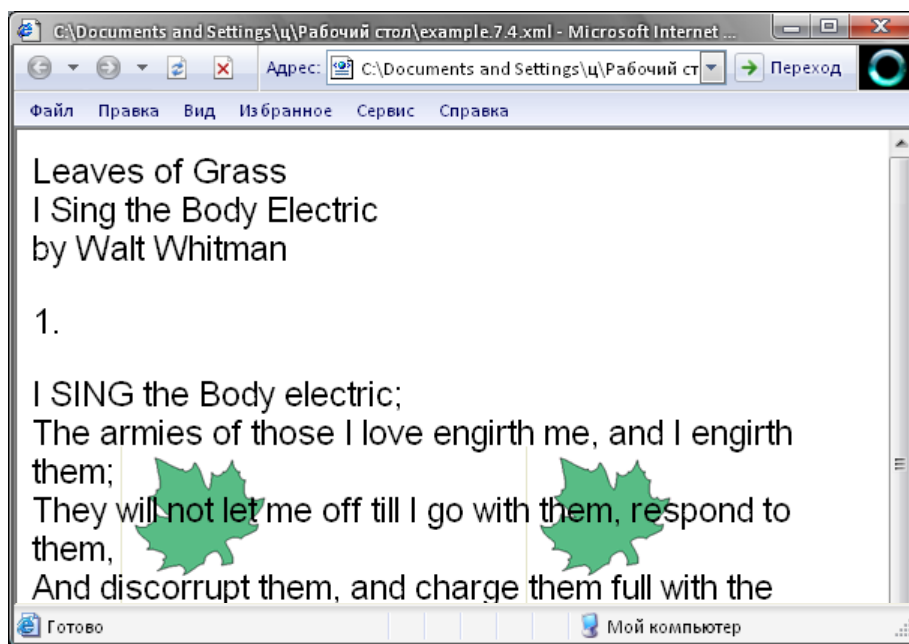


Рис. 7.10. Отображение в Internet Explorer 5 страницы, для которой применено свойство background-repeat

2) repeat-x. Повторяет изображение только в горизонтальном направлении. Например, следующее правило STANZA приведет к отображению документа, как показано на рис. 7.11:

STANZA

```
{background-image:url(Leaf.bmp);  
background-repeat:repeat-x}
```

3) repeat-y. Повторяет изображение только в вертикальном направлении. Например, следующее правило STANZA отобразит документ, как это показано на рис. 7.12:

STANZA

```
{background-image:url(Leaf.bmp);  
background-repeat:repeat-y}
```

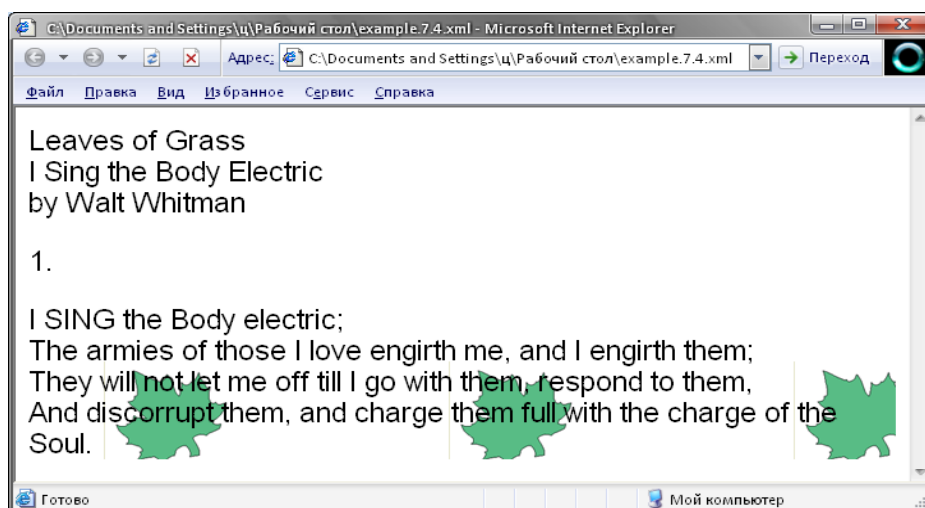


Рис. 7.11. Отображение в Internet Explorer 5 изображения в горизонтальном направлении

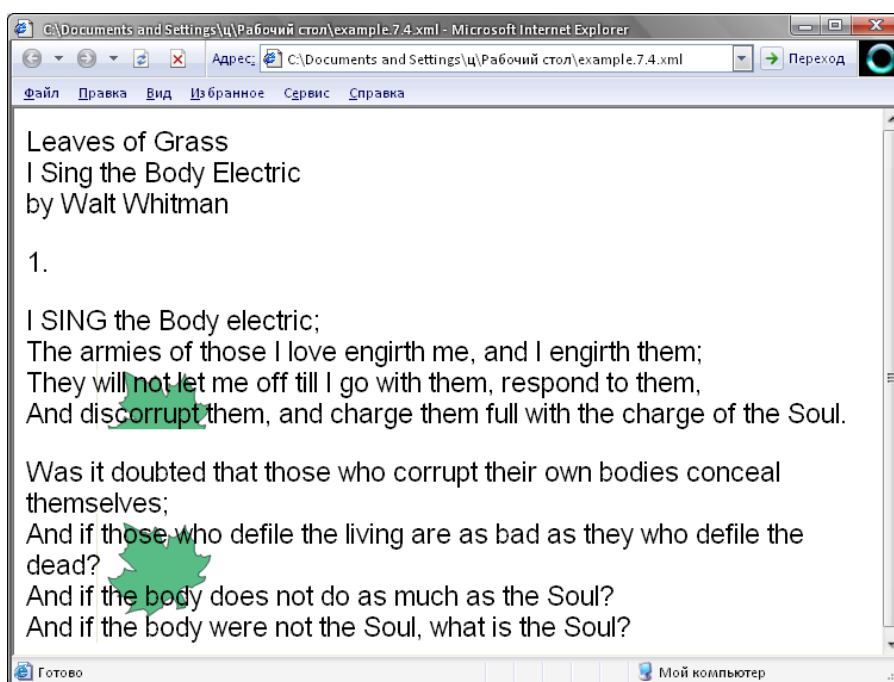


Рис. 7.12. Отображение в Internet Explorer 5 изображения в вертикальном направлении

4) no-repeat. Приводит к однократному отображению рисунка. Например, следующее правило STANZA приведет к отображению документа, как показано на рис. 7.13:

```
STANZA
{background-image:url(Leaf.bmp);
background-repeat:no-repeat}
```

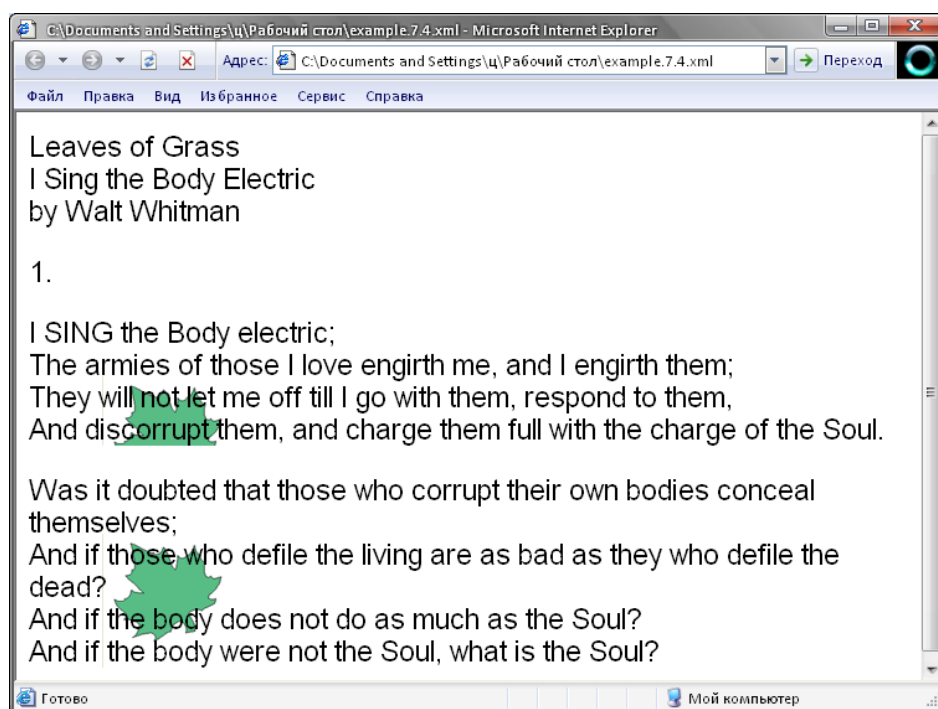


Рис. 7.13. Отображение в Internet Explorer 5 страницы с однократным изображением рисунка

7.5.4. Установка свойства background-position

По умолчанию верхний левый угол фонового изображения (или верхний левый угол верхней левой копии изображения, если оно повторяется) совмещается с верхним левым углом элемента. Вы можете изменить такое совмещение, задав значение для свойства background-position. Вы можете назначить этому свойству три различных вида значений:

– значения горизонтальной и вертикальной позиции в единицах размеров. Вы можете задать свойству background-position два значения размеров. Первое значение определяет горизонтальную позицию изображения внутри элемента, а второе значение – вертикальную позицию изображения внутри элемента. Вы можете присвоить любой тип численного значения размера. Например, следующее правило помещает верхний левый угол изображения на 5 дюймов правее и на 25 дюймов ниже верхнего левого угла элемента STANZA:

```
STANZA
{background-image:url(Leaf.bmp);
background-repeat:no-repeat;
background-position:.5in .25in}
```

На рис. 7.14 показано, как будет выглядеть результат.

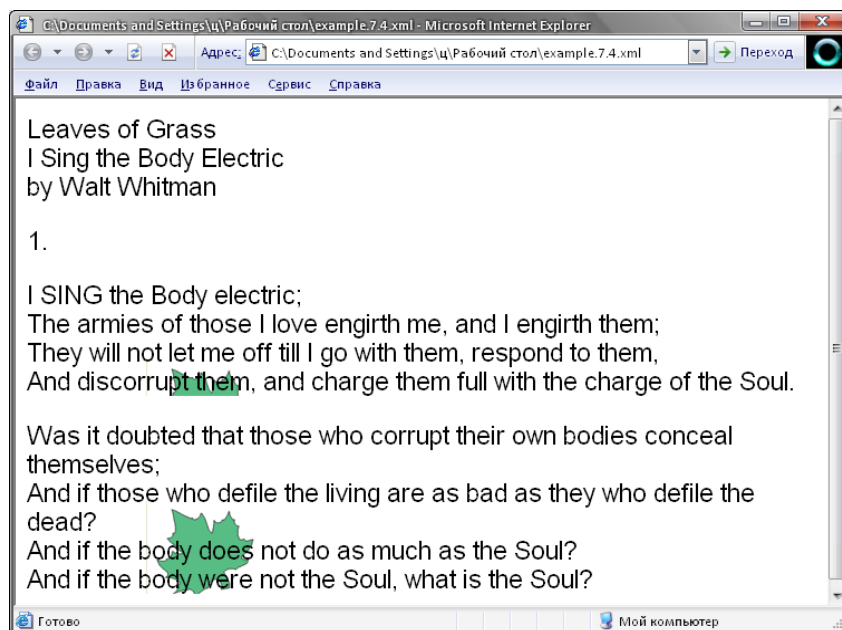


Рис. 7.14. Отображение в Internet Explorer 5 перемещения элементов STANZA

Если изображение будет повторяться в соответствии с приведенным ниже правилом, весь узор, состоящий из повторных изображений, окажется смещенным (рис. 7.15).

```
STANZA
{background-image:url(Leaf.bmp);
background-repeat:repeat;
background-position:.5in .25in}
```

– значения горизонтальной и вертикальной позиции в процентах. Вы можете установить для свойства `background-position` два процентных значения. Первое из них определяет горизонтальную позицию изображения внутри элемента, при этом 0% соответствует левому краю (горизонтальная позиция по умолчанию), 50% – центру элемента по горизонтали, а 100% – правому краю элемента. Второе значение в процентах определяет вертикальную позицию изображения, при этом 0% соответствует верхнему краю (вертикальная позиция по умолчанию), 50% – центру элемента по вертикали, а 100% – нижнему краю элемента.

Например, следующее правило поместит изображение в середину элемента:

```
STANZA
{background-image:url(Leaf.bmp);
background-repeat:no-repeat;
background-position:50% 50%}
```

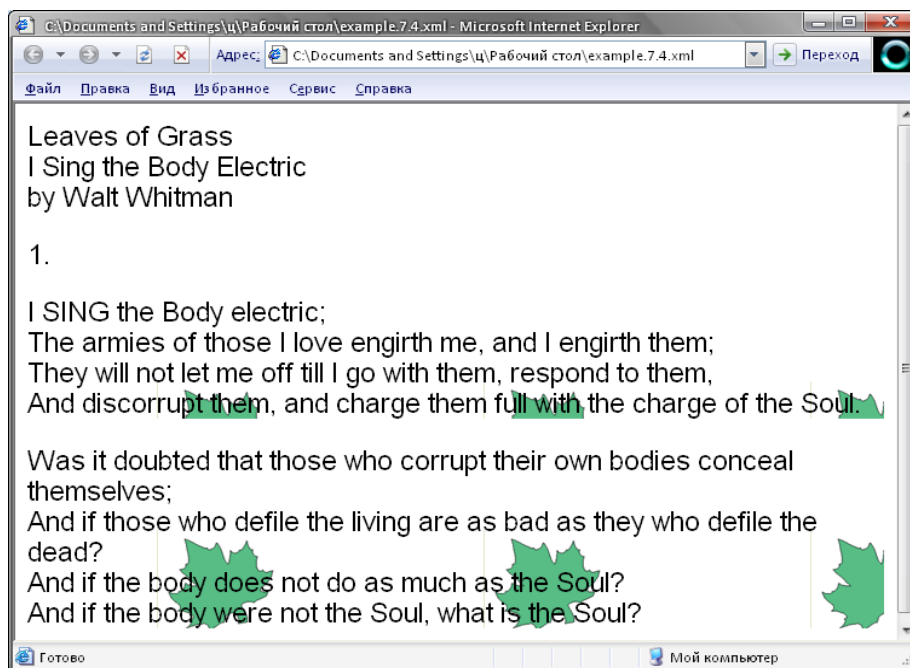


Рис. 7.15. Отображение в Internet Explorer 5 смещения изображения

На рис. 7.16 представлен результат действия правила STANZA.

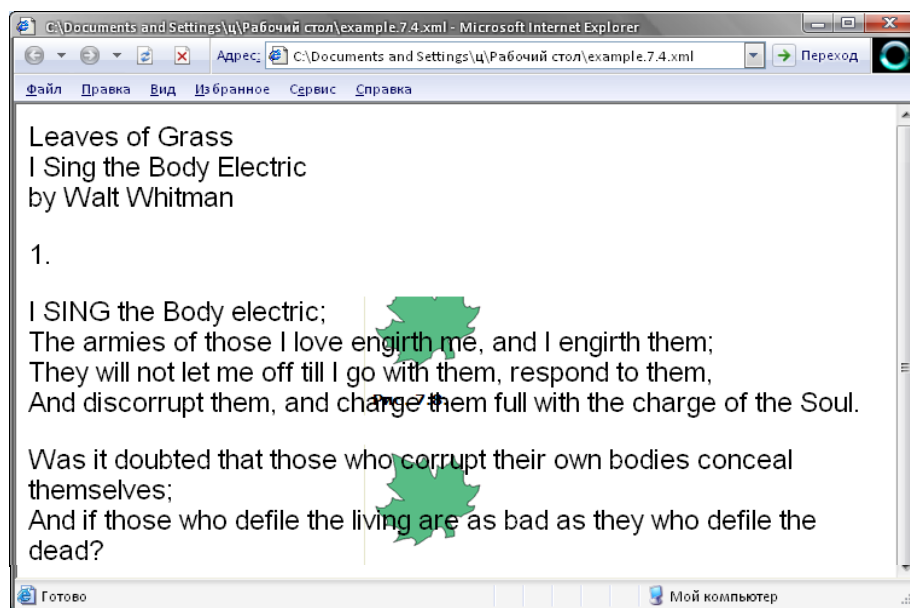


Рис. 7.16. Отображение в Internet Explorer 5 помещения изображения в середину

Если изображение повторяется в соответствии со следующим правилом, весь узор из повторяющихся изображений будет смещен на заданную величину (рис. 7.17):

STANZA

```
{background-image:url(Leaf.bmp);  
background-repeat:repeat;  
background-position:50% 50%}
```

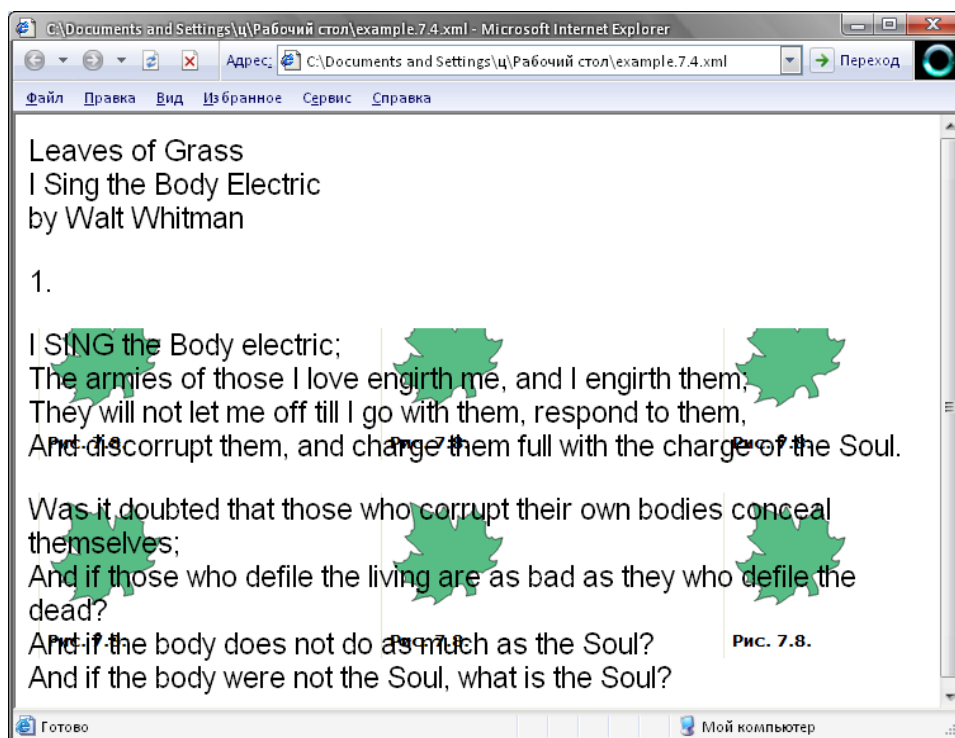


Рис. 7.17. Отображение в Internet Explorer 5 смещения изображения на заданную величину

– значения в виде ключевых слов. Вы можете задать позицию фонового изображения, назначив свойству background-position одно или два ключевых слова CSS.

Например, введите ключевые слова right и bottom, как в следующем правиле, и ваше изображение будет помещено в правый нижний угол элемента (рис. 7.18):

STANZA

```
{background-image:url(Leaf.bmp);  
background-repeat:no-repeat;  
background-position:right bottom}
```

На рис. 7.19 приведены различные варианты сочетаний ключевых слов и соответствующие им позиции изображения.

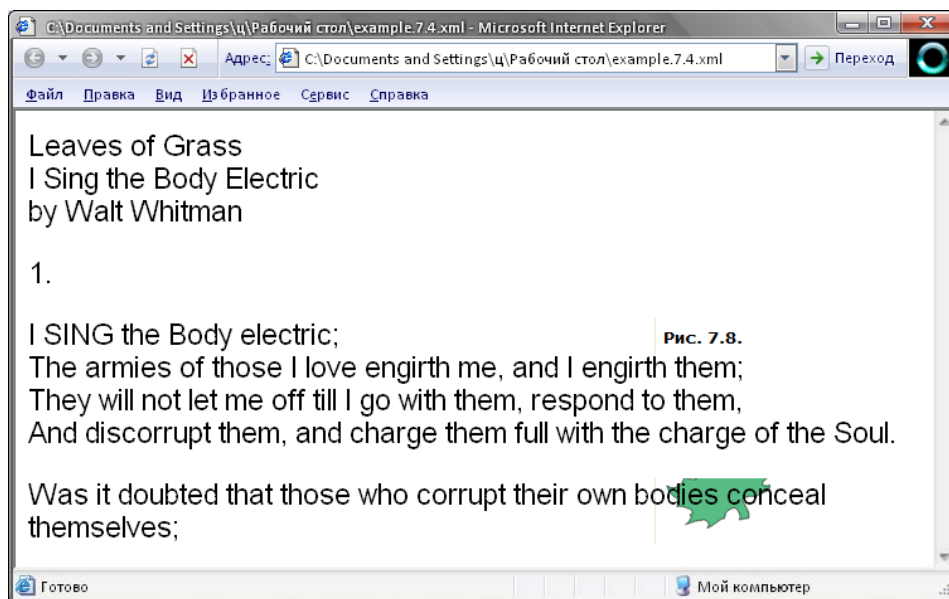


Рис. 7.18. Отображение в Internet Explorer 5 смещения изображения в правый нижний угол

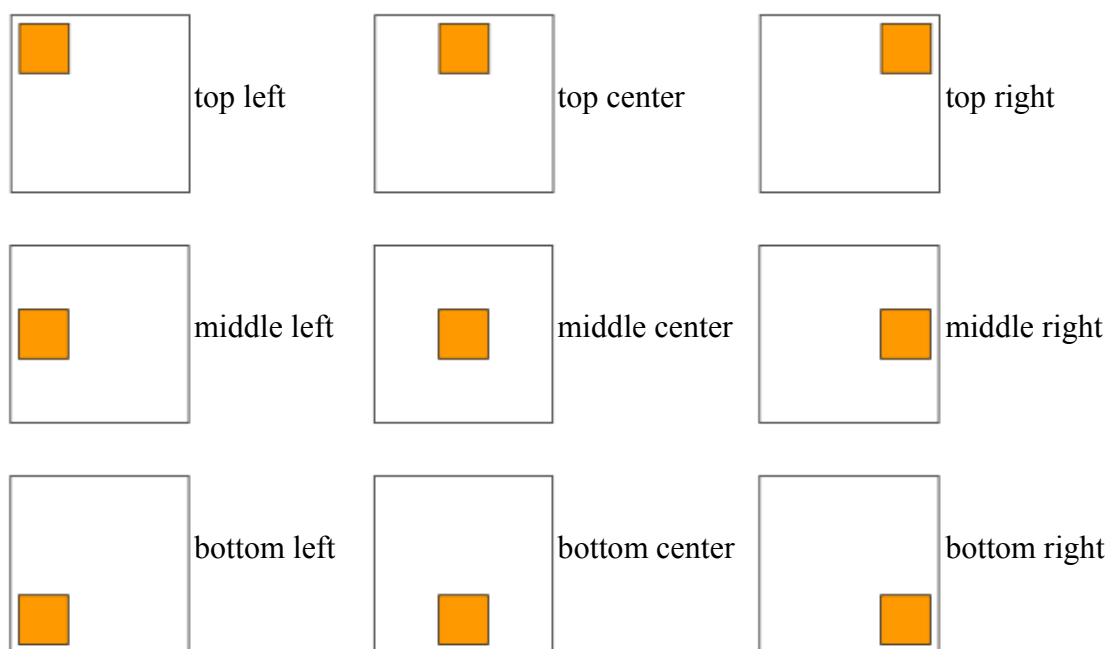


Рис. 7.19. Варианты сочетаний ключевых слов и соответствующие им позиции изображения

Порядок ключевых слов значения не имеет. Например, background-position:bottom right эквивалентно background-position:right bottom, background-position:top center эквивалентно background-position:center top и т. д.

7.6. УСТАНОВКА СВОЙСТВ РАЗБИВКИ ТЕКСТА И ВЫРАВНИВАНИЯ

Стандарт CSS поддерживает следующие свойства, позволяющие модифицировать разбивку, выравнивание и другие характеристики текста:

- 1) letter-spacing;
- 2) vertical-align;
- 3) text-align;
- 4) text-indent;
- 5) line-height;
- 6) text-transform;
- 7) text-decoration.

Дочерние элементы наследуют все эти свойства, за исключением vertical-align.

7.6.1. Установка свойства letter-spacing

Вы можете воспользоваться свойством letter-spacing для увеличения или уменьшения просвета между символами в тексте элемента. Для увеличения просвета свойству letter-spacing следует присвоить положительное значение в соответствующих единицах размера. Например, данное правило увеличивает просвет между символами на одну четверть от высоты текста:

```
TITLE {letter-spacing:.25em}
```

Вы можете задать свойству letter-spacing отрицательное значение для уменьшения просвета между символами на значение в соответствующих единицах размера. Например, это правило уменьшает просвет между символами на половину пункта:

```
TITLE {letter-spacing:-.5pt}
```

Вы также можете выбрать обычную величину просвета, установив для свойства letter-spacing значение normal. Например, следующая таблица стилей, присоединенная к XML-документу из листинга 7.4 (см. на с. 159), назначает увеличенный просвет между символами для элемента TITLE и обычный просвет между символами для элемента SUBTITLE (второе назначение необходимо, чтобы отменить увеличенный просвет между символами, который в противном случае элемент SUBTITLE унаследовал бы от родительского элемента TITLE):


```

POEM
    {font-size:145%}
POEM, TITLE, SUBTITLE, AUTHOR, SECTION, NUMBER, STANZA, VERSE
    {display:block}
SECTION, STANZA
    {margin-top:1em}
TITLE
    {letter-spacing:.5em}
SUBTITLE
    {letter-spacing:normal}

```

В соответствии с правилами этой таблицы стилей Internet Explorer 5 отобразит XML-документ, как показано на рис. 7.20.

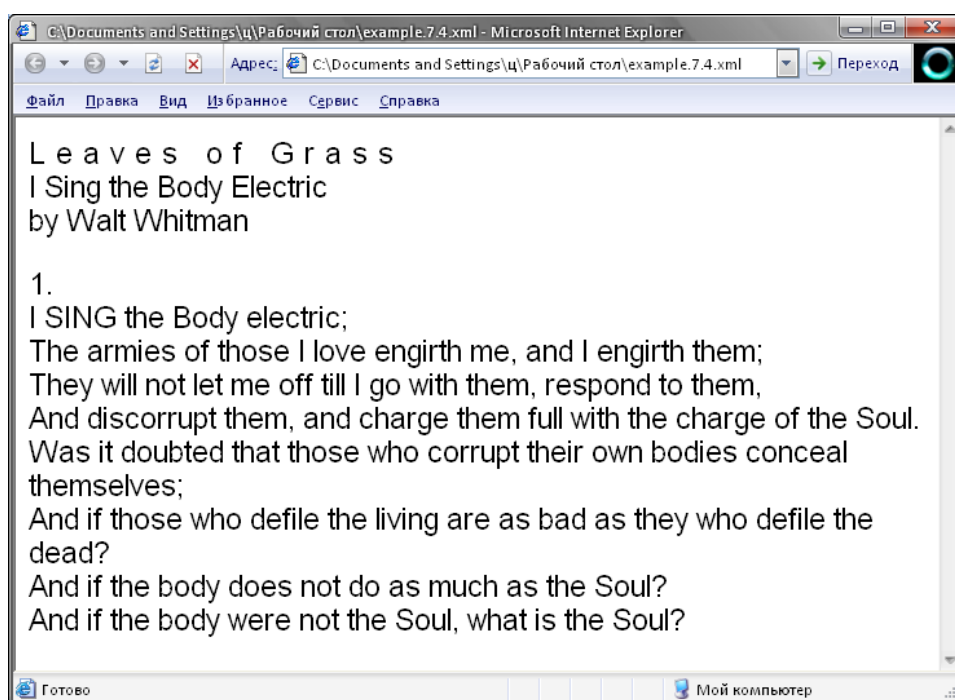


Рис. 7.20. Отображение в Internet Explorer 5 результата применения свойства letter-spacing

7.6.2. Установка свойства vertical-align

Вы можете использовать свойство vertical-align для создания верхних или нижних индексов (надстрочных или подстрочных символов). Это свойство оказывает воздействие только на элементы inline.

Вы можете назначить свойству vertical-align одно из ключевых слов CSS, описанных в табл. 7.10. Чтобы получить образцы текста, присвоим значение свойству vertical-align только для элемента

CHILD, который является элементом inline и описан в документе следующим образом:

```
<PARENT>PARENT ELEMENT  
  <CHILD>CHILD ELEMENT</CHILD>  
</PARENT>
```

Дочерние элементы не наследуют свойство vertical-align (см. табл. 7.10).

Таблица 7.10

Ключевые слова для свойства vertical-align

Ключевое слово для vertical-align	Пример правила CSS	Описание
baseline (по умолчанию)	CHILD (font-size:75%; Vertical-align:baseline)	Совмещает базовую линию текста элемента с базовой линией текста родительского элемента
sub	CHILD (font-size:75%; Vertical-align:sub)	Отображает текст элемента как нижний индекс
super	CHILD (font-size:75%; Vertical-align:super)	Отображает текст элемента как верхний индекс

7.6.3. Установка свойства text-align

Вы можете воспользоваться свойством text-align для управления горизонтальным выравниванием текста элемента. Это свойство будет работать только в том случае, если вы применяете его для элемента типа block. Оно воздействует как на сам элемент, так и на дочерние элементы, которые он содержит, независимо от того, относятся ли они к типу block или inline.

Свойство text-align воздействует на выравнивание текста внутри области содержимого текста. По умолчанию область содержимого текста занимает практически полную ширину окна браузера. Однако вы можете модифицировать как ширину, так и положение области текста элемента.

Вы можете назначить свойству text-align одно из следующих трех ключевых слов:

- left. Выравнивает каждую строку по левому краю. Предположим, вы применяете следующее правило к XML-документу из листинга 7.4 (в добавление к остальным правилам, содержащимся

в таблице стилей из листинга 7.3 (см. на с. 158), за исключением установки свойства background-image, которая удалена):

```
ПОЕМ {text-align:left}
```

Поэма будет выглядеть, как показано на рис. 7.21.

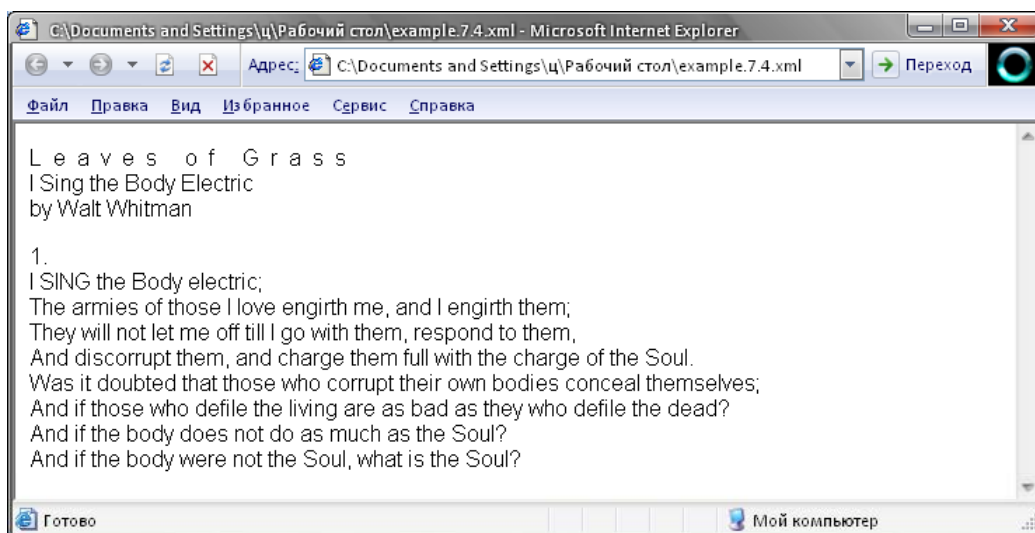


Рис. 7.21. Отображение в Internet Explorer 5 результата выравнивания текста по левому краю

– right. Выравнивает каждую строку по правому краю. Так, следующее правило выравнивает текст поэмы вправо (рис. 7.22):

```
ПОЕМ {text-align:right}
```

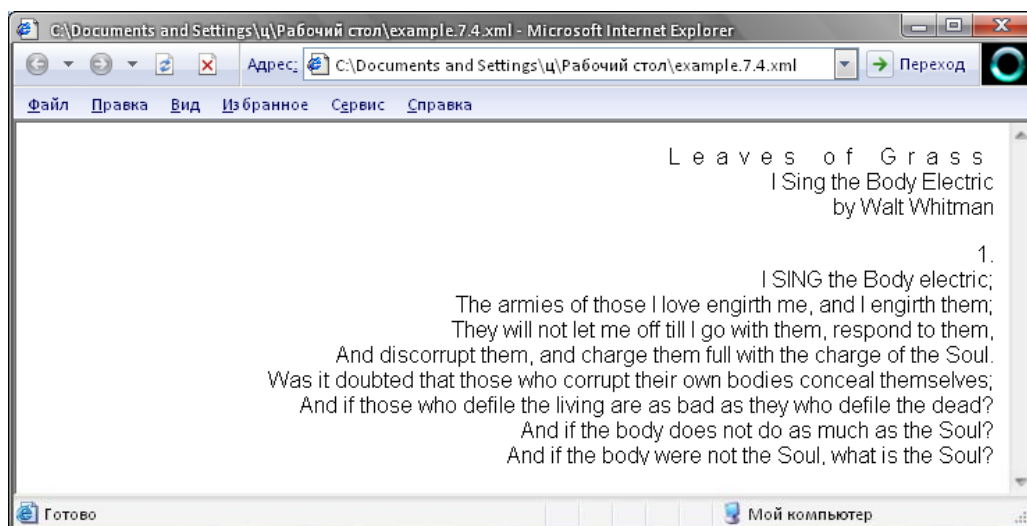


Рис. 7.22. Отображение в Internet Explorer 5 результата выравнивания текста по правому краю

– center. Центрирует строки по горизонтали. Например, следующее правило центрирует весь текст поэмы, как показано на рис. 7.23:

```
POEM {text-align:center}
```

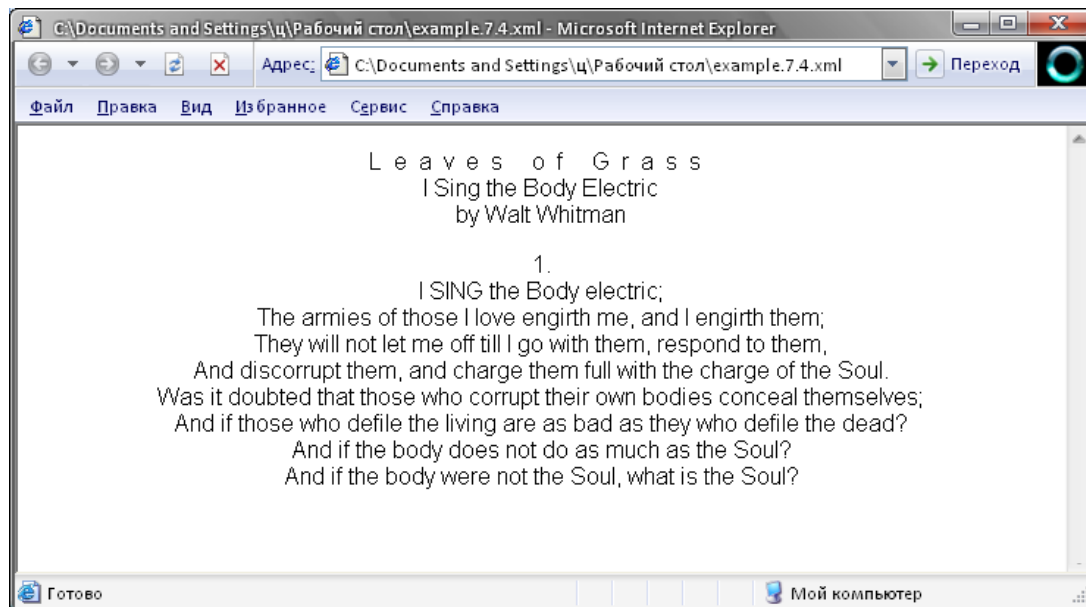


Рис. 7.23. Отображение в Internet Explorer 5 результата выравнивания текста по центру

7.6.4. Установка свойства text-indent

Вы можете воспользоваться свойством text-indent, чтобы задать отступ для первой строки текста элемента. Вы можете устанавливать для свойства text-indent все виды значений размеров, описанных ранее в 7.3.4 «Задание значений в размерных единицах». Например, следующее правило задает отступ для первой строки элемента VERSE, равный трехкратной высоте ее шрифта:

```
VERSE {text-indent:3em}
```

Вот как будет выглядеть элемент VERSE:

It is in his walk, the carriage of his
neck, the flex of his waist and knees-dress
does not hide him;

Альтернативой является установление величины отступа в процентах от полной ширины текста элемента. Например, это правило

задает смещение первой строки элемента VERSE на половину ширины элемента:

```
VERSE {text-indent:50%}
```

Элемент VERSE будет выглядеть следующим образом:

It is in his walk, the
carriage of his neck, the flex of his waist
and knees-dress does not hide him;

|| *Совет.* Чтобы задать отступ для всех строк элемента (а не только для первой строки), воспользуйтесь свойством `margin-left`.

Вы можете установить отрицательное значение (в размерных единицах или в процентах), чтобы сдвинуть первую строку влево относительно других строк. Однако если вы просто присвоите отрицательное значение свойству `text-indent`, первая часть строки окажется скрытой (рис. 7.24).

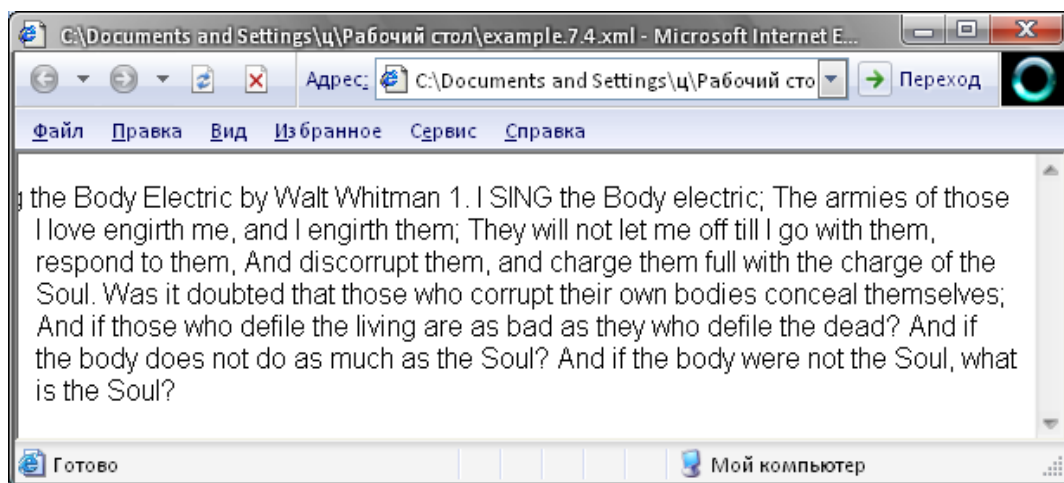


Рис. 7.24. Отображение в Internet Explorer 5 результата применения свойства `text-indent`

Чтобы избежать этого, вы должны задать для элемента левое поле. Например, следующее правило устанавливает левое поле шириной в 4em (`margin-left:4em`), а затем сдвигает первую строку на 2em (`text-indent:-2em`), создавая висячий отступ (рис. 7.25):

```
VERSE  
{margin-left:4em;  
text-indent:-2em}
```

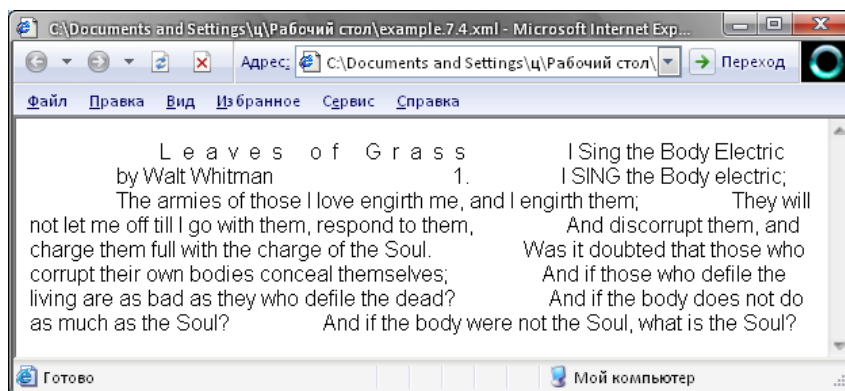


Рис. 7.25. Отображение в Internet Explorer 5 результата применения свойств margin-left:4em и text-indent:-2em

7.6.5. Установка свойства line-height

Свойство line-height управляет расстоянием между базовыми линиями соседних строк текста элемента. Вы можете воспользоваться этим свойством для разбивки строк по вертикали.

Свойству line-height можно присвоить значение, которое выражено в любой из размерных единиц, описанных ранее в 7.3.4 «Задание значений в размерных единицах». Предположим, вы применили следующее правило для XML-документа из листинга 7.4:

```
STANZA {line-height:2em}
```

Текст элемента STANZA будет иметь двойной междустрочный интервал, как показано на рис. 7.26.

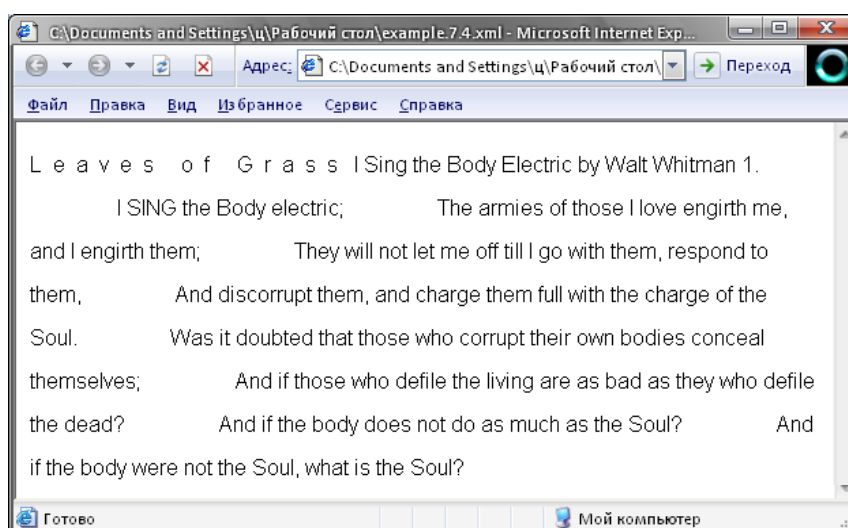


Рис. 7.26. Отображение в Internet Explorer 5 результата применения свойства line-height

Альтернативой является задание междустрочного интервала в процентах от высоты текста элемента:

```
STANZA {line-height:200%}
```

7.6.6. Установка свойства text-transform

Вы можете воспользоваться свойством text-transform для управления стилем прописных букв текста элемента при отображении его браузером. Вы можете установить для свойства text-transform значения в виде ключевых слов, описанных в табл. 7.11.

Таблица 7.11

Ключевые слова для свойства text-transform

Ключевое слово для text-transform	Пример правила CSS	Описание
uppercase	STANZA {text-transform:uppercase}	Преобразует в прописные все буквы
lowercase	STANZA {text-transform:lowercase}	Отображает все буквы строчными
none	STANZA {text-transform:none}	Отображает текст без изменения стиля прописных букв

7.6.7. Установка свойства text-decoration

Вы можете использовать свойство text-decoration для рисования различных типов линий внутри текста элемента. В табл. 7.12 представлены ключевые слова, которые могут быть назначены этому свойству.

Таблица 7.12

Ключевые слова для свойства text-decoration

Ключевое слово для text-decoration	Пример правила CSS	Описание
underline	TITLE {text-decoration:underline}	Рисует линию под текстом
overline	TITLE {text-decoration:overline}	Рисует линию над текстом
line-through	TITLE {text-decoration:line-through}	Рисует линию, перечеркивающую текст
none	TITLE {text-decoration:none}	Отображает текст без линий

Вы можете применить более одного типа линий, присвоив свойству `text-decoration` несколько значений. (Однако включение ключевого слова `none` отменяет действие всех ранее установленных ключевых слов.) Например, следующее правило указывает браузеру нарисовать линию поверх и под текстом:

```
TITLE {text-decoration:underline overline}
```

7.7. УСТАНОВКА СВОЙСТВ ТЕКСТОВЫХ ОБЛАСТЕЙ

Спецификация CSS поддерживает ряд свойств (типа `box`), которые вы можете использовать для форматирования блока текста, принадлежащего элементу. К этим свойствам относятся (рис. 7.27):

- 1) свойство `margin` добавляет невидимое (прозрачное) поле вокруг элемента, снаружи от видимой рамки;
- 2) свойство `border` отображает видимую рамку – определенного стиля – вокруг элемента, снаружи от просвета (если он имеется);
- 3) свойство `padding` добавляет просвет непосредственно снаружи от содержимого элемента, но внутри рамки (если она имеется);
- 4) свойства задания размеров `height` и `width` устанавливают размеры области содержимого элемента с добавленными просветом и рамкой;
- 5) свойства задания позиций `float` и `clear` устанавливают положение элемента относительно окружающих элементов.

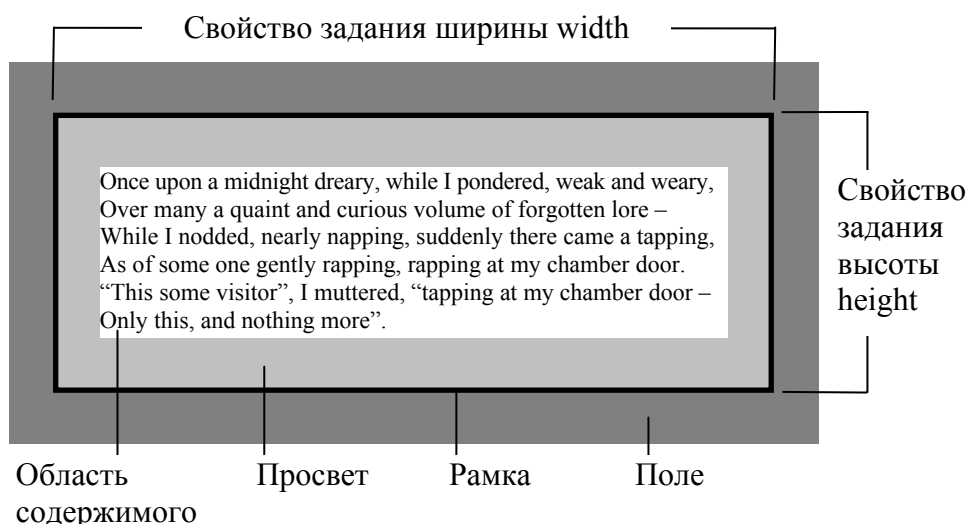


Рис. 7.27. Свойства CSS

Элемент `block` – это элемент для свойства `display` с установленным значением `block`, а элемент `inline` – это элемент для свойства `display` с установленным значением `inline`. В Internet Explorer 5.0 и 5.01 первые три группы свойств (`margin`, `border` и `padding`) действуют только на элементы `block`. Однако в Internet Explorer 5.5 вы можете использовать эти свойства как для элементов `block`, так и для элементов `inline`.

7.8. УСТАНОВКА СВОЙСТВ УПРАВЛЕНИЯ ПОЛЯМИ

По умолчанию ширина полей вокруг элемента равна нулю. Чтобы добавить поле с одной или с нескольких сторон элемента, вы можете присвоить ненулевое значение одному или нескольким из следующих свойств:

- `margin-top`;
- `margin-right`;
- `margin-bottom`;
- `margin-left`.

Вы можете устанавливать для этих свойств любые значения в размерных единицах, описанных ранее в 7.3.4 «Задание значений в размерных единицах». Например, следующее правило добавляет поле слева и справа от элемента `STANZA`. Ширина поля в 2 раза больше высоты текста элемента:

```
STANZA
{margin-left:2em;
margin-right:2em}
```

Вы также можете устанавливать величину поля в процентах относительно ширины родительского элемента. Например, следующее правило создает левое поле, равное одной четвертой от ширины родительского элемента:

```
STANZA {margin-left:25%}
```

Вы можете назначать одинаковые величины полей по всем четырем сторонам элемента путем присвоения одного значения (в размерных единицах или в процентах) свойству `margin`. В качестве примера возьмем таблицу стилей, которая представлена в листинге 7.5 и присоединена к XML-документу, приведенному

в листинге 7.6. В соответствии с этой таблицей стилей текст отображается без полей (рис. 7.28).

Листинг 7.5. Raven.css

```
/* File Name: Raven.css */
POEM
    {font-size:small}
POEM, TITLE, AUTHOR, DATE, STANZA, VERSE
    {display:block}
```

Листинг 7.6. Raven.xml

```
<?xml version="1.0"?>
<!-- File Name: Raven.xml -->
<?xml-stylesheet type="text/css" href="Raven.css"?>
<POEM>
<TITLE>The Raven</TITLE>
<AUTHOR>Edgar Allan Poe</AUTHOR>
<DATE>1845</DATE>
<STANZA>
    <VERSE>Once upon a midnight dreary, while I
pondered, weak and weary,</VERSE>
    <VERSE>Over many a quaint and curious volume of
forgotten lore&#8212;</VERSE>
    <VERSE>While I nodded, nearly napping, suddenly
there came a tapping,</VERSE>
    <VERSE>As of some one gently rapping, rapping at
my chamber door.</VERSE>
    <VERSE>"'Tis some visitor," I muttered, "tapping
at my chamber door&#8212;</VERSE>
    <VERSE>Only this, and nothing more."</VERSE>
</STANZA>
<STANZA>
    <VERSE>Ah, distinctly I remember it was in the
bleak December,</VERSE>
    <VERSE>And each separate dying ember wrought its
ghost upon the floor.</VERSE>
    <VERSE>Eagerly I wished the morrow;&#8212;vainly I
had sought to borrow</VERSE>
    <VERSE>From my books surcease of
sorrow&#8212;sorrow for the lost Lenore&#8212;</VERSE>
    <VERSE>For the rare and radiant maiden whom the
angels name Lenore&#8212;</VERSE>
    <VERSE>Nameless here for evermore.</VERSE>
</STANZA>
</POEM>
```

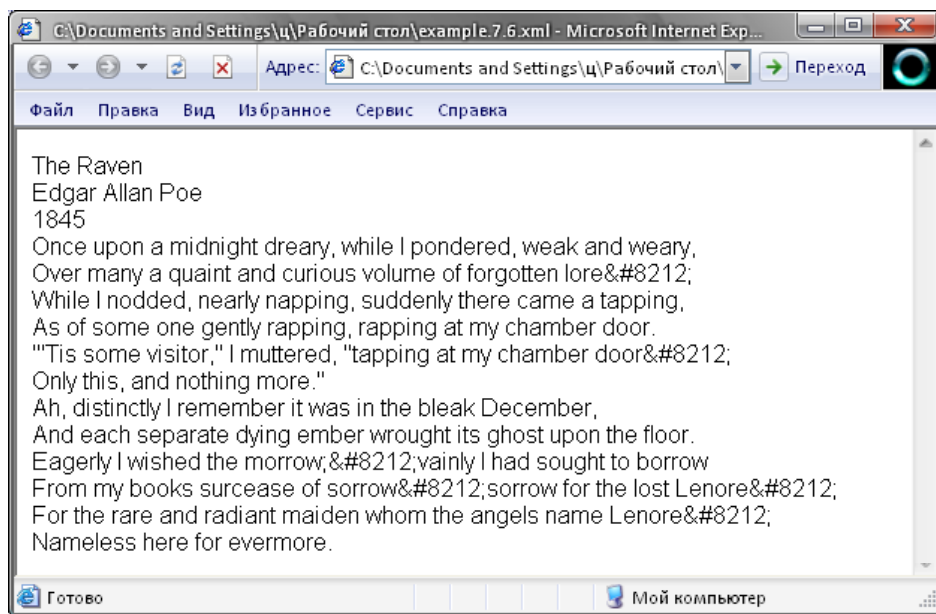


Рис. 7.28. Отображение в Internet Explorer 5 текста без полей

Добавив следующее правило в таблицу стилей, которое устанавливает поля в 2,5em по всем четырём сторонам элементов STANZA, вы получите то, что представлено на рис. 7.29:

```
STANZA {margin:2.5em}
```

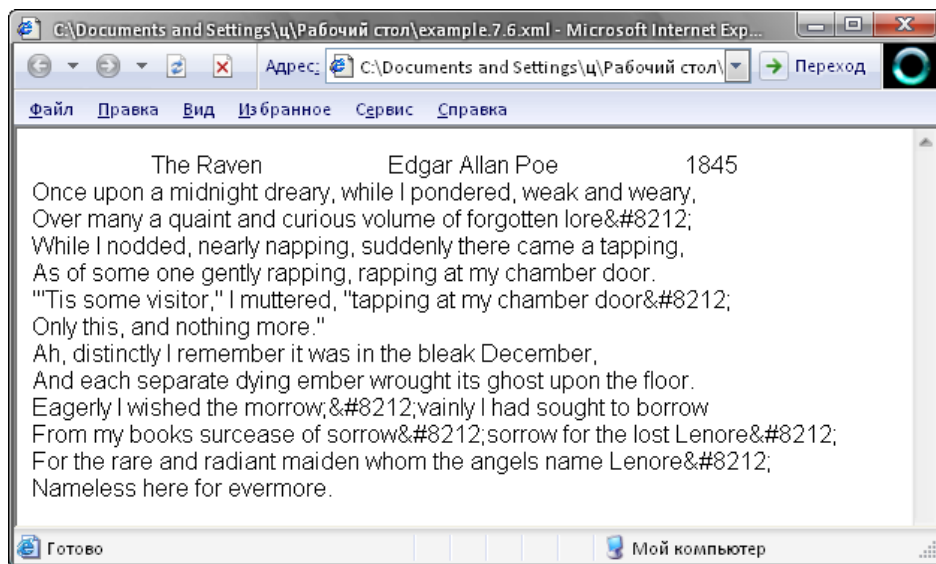


Рис. 7.29. Отображение в Internet Explorer 5 текста с полем в 2,5em

⇒ **Примечание.** Зона полей всегда прозрачна, т. е. через нее видна фоновая заливка или фоновый рисунок родительского элемента (или браузера).

7.9. УСТАНОВКА СВОЙСТВ УПРАВЛЕНИЯ ОБРАМЛЕНИЕМ

Вы можете воспользоваться следующими свойствами, чтобы создавать видимое обрамление вокруг элемента:

- 1) border-style;
- 2) border-width;
- 3) border-color.

7.9.1. Установка свойства border-style

Вы можете использовать свойство border-style для добавления видимого обрамления с одной или с нескольких сторон элемента, а также для установки стиля рамки. Вы можете назначать свойству border-style ключевые слова, представленные в табл. 7.13.

Таблица 7.13

Ключевые слова для свойства border-style

Ключевое слово для border-style	Пример правила CSS
solid	TITLE {border-style:solid}
dotted (только в Internet Explorer 5.5)	TITLE {border-style:dotted}
dashed (только в Internet Explorer 5.5)	TITLE {border-style:dashed}
double	TITLE {border-style:double}
groove	TITLE {border-style:groove}
ridge	TITLE {border-style:ridge}
inset	TITLE {border-style:inset}
outset	TITLE {border-style:outset}
none	TITLE {border-style:none}

Вы можете использовать разнообразные стили рамки с каждой стороны элемента, назначая различные ключевые слова свойству border-style. Порядок значений соответствует рамке сверху, справа, снизу и слева. Так, следующее правило отображает сплошную рамку сверху и снизу элемента TITLE, без рамок слева и справа:

```
TITLE {border-style:solid none solid none}
```

В другом примере добавление следующего правила в таблицу стилей, представленную в листинге 7.5, приводит к отображению рамки со всех сторон каждого из элементов STANZA

в XML-документе из листинга 7.6. Снаружи границ рамки имеются поля размером 2,5em:

STANZA

```
{margin:2.5em;  
border-style:double solid double solid}
```

Нижняя и верхняя границы являются двойными, а левая и правая – одинарными, как показано на рис. 7.30.

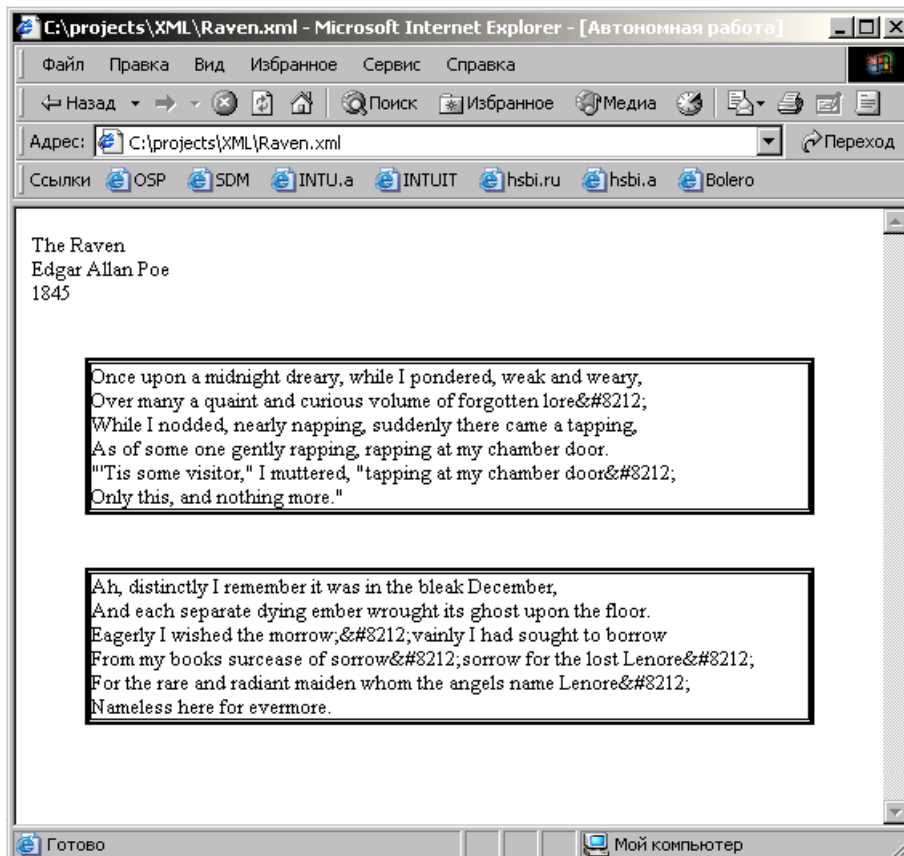


Рис. 7.30. Отображение в Internet Explorer 5 результата применения свойства border-style

7.9.2. Установка свойства border-width

Если вы установили видимые линии рамки с одной или с нескольких сторон элемента с использованием свойства border-style, вы можете изменять толщину рамки путем задания ключевых слов свойству border-width (табл. 7.14).

Альтернативой является задание толщины рамки путем присвоения свойству border-width значений в единицах размеров,

описанных ранее в 7.3.4 «Задание значений в размерных единицах». Например, следующее правило устанавливает для элемента TITLE рамку толщиной 1 пиксель (наименьшая толщина, которая может быть отображена на мониторе) по всем сторонам:

```
TITLE
{border-style:solid;
border-width:1px}
```

Таблица 7.14

Ключевые слова для свойства border-width

Ключевое слово для border-width	Пример правила CSS
thin	TITLE {border-style:solid; border-width:thin}
medium	TITLE {border-style:solid; border-width:medium}
thick	TITLE {border-style:solid; border-width:thick}

Вы можете задавать различную толщину рамки с каждой стороны элемента путем присвоения четырех различных значений – ключевых слов или значений в единицах размера – свойству border-width. Порядок следования значений соответствует верхней, правой, нижней и левой границам рамки. Например, добавление следующего правила в таблицу стилей из листинга 7.5 создает сплошную одинарную рамку со всех сторон элемента STANZA:

```
STANZA
{margin:2.5em;
border-style:solid;
border-width:1px thick 1px thick}
```

Границы рамки сверху и снизу имеют минимальную толщину, в то время как левая и правая границы имеют увеличенную толщину, как показано на рис. 7.31.

7.9.3. Установка свойства border-color

По умолчанию рамки, которые вы создаете с использованием свойства border-style, имеют тот же цвет, что и текущий цвет, установленный для свойства color элемента. Вы можете менять цвет для всех четырех границ рамки путем присвоения свойству border-color любого из значений цвета, описанных в пункте «Задание значений цвета» (см. на с. 156–157) ранее в этой главе.

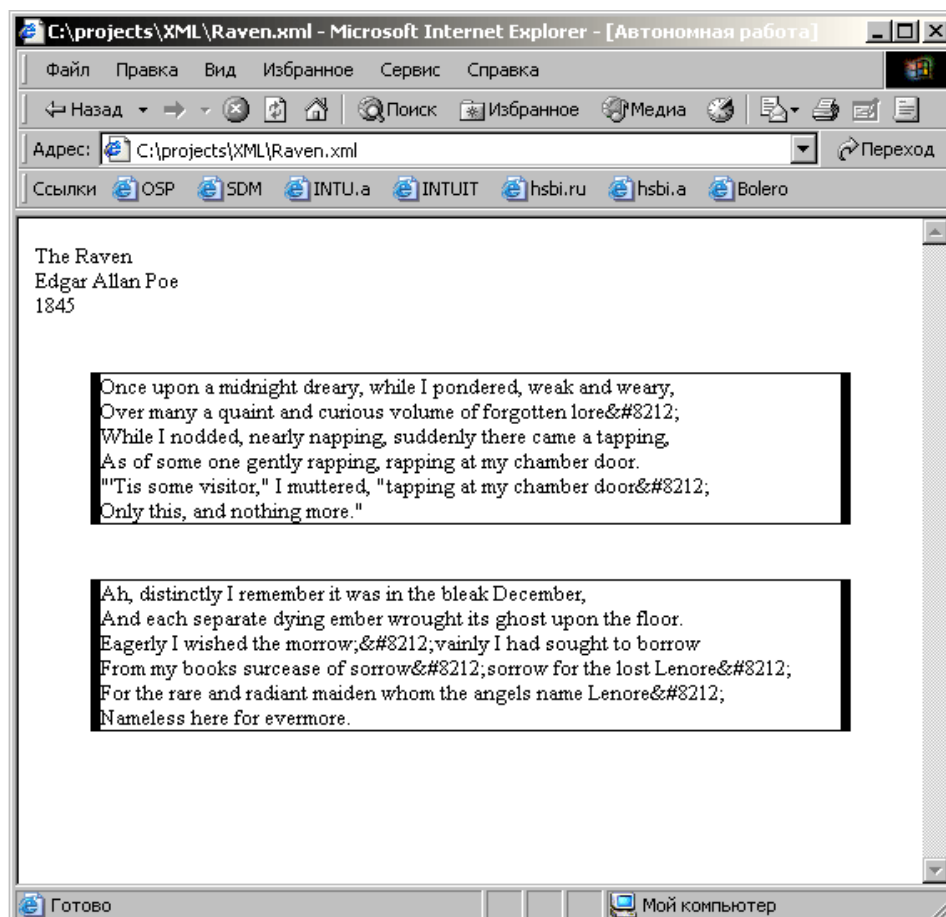


Рис. 7.31. Отображение в Internet Explorer 5 результата применения свойства border-color

Например, следующее правило добавляет сплошную рамку красного цвета со всех сторон элемента TITLE:

```
TITLE
{border-style:solid;
border-color:red}
```

Вы также можете задавать различный цвет для отдельных границ рамки вокруг элемента, присваивая четыре различных значения цвета свойству border-color. Например, следующее правило устанавливает сплошные красные линии рамки сверху и снизу элемента TITLE, а также сплошные зеленые линии рамки слева и справа от элемента:

```
TITLE
{border-style:solid;
border-color:red green red green}
```

7.10. УСТАНОВКА СВОЙСТВ ПРОСВЕТА МЕЖДУ ОБРАМЛЕНИЕМ И ТЕКСТОМ

Напомним, что свойства просвета (padding) добавляют пространство непосредственно вокруг содержимого элемента внутри от имеющейся вокруг элемента рамки. Без просвета границы рамки располагаются непосредственно вблизи текста элемента. Добавление просвета улучшает восприятие рамки.

По умолчанию ширина просвета для элемента устанавливается близкой к нулю. Чтобы добавить просвет с одной или с нескольких сторон от текста элемента, вы можете присвоить ненулевое значение одному или нескольким из следующих свойств:

- padding-top;
- padding-right;
- padding-bottom;
- padding-left.

Вы можете устанавливать для этих свойств значения в любых размерных единицах, описанных в 7.3.4 «Задание значений в размерных единицах». Например, следующее правило добавляет просвет сверху и снизу от элемента STANZA. Ширина просвета равна двойной высоте текста элемента:

```
STANZA
  {padding-top:2em;
   padding-bottom:2em}
```

Вы также можете задавать ширину просвета в процентах относительно ширины родительского элемента. Так, следующее правило добавляет просвет слева от элемента STANZA. Ширина просвета составляет одну четвертую от ширины родительского элемента:

```
STANZA {padding-left:25%}
```

Помните, что вы можете добавлять просвет одинаковой ширины со всех четырех сторон элемента, присвоив единственное значение (в размерных единицах или в процентах) свойству padding. Допустим, что вы добавили следующее правило в таблицу стилей, содержащуюся в листинге 7.5:

```
STANZA
  {margin:2.5em;
   border-style:solid;
   padding:2em}
```


Это правило задает следующее форматирование областей для каждого элемента STANZA:

- 1) просвет шириной 2em непосредственно вокруг элемента;
- 2) обрамление сплошной линией по контуру просвета;
- 3) поле снаружи от рамки (рис. 7.32).

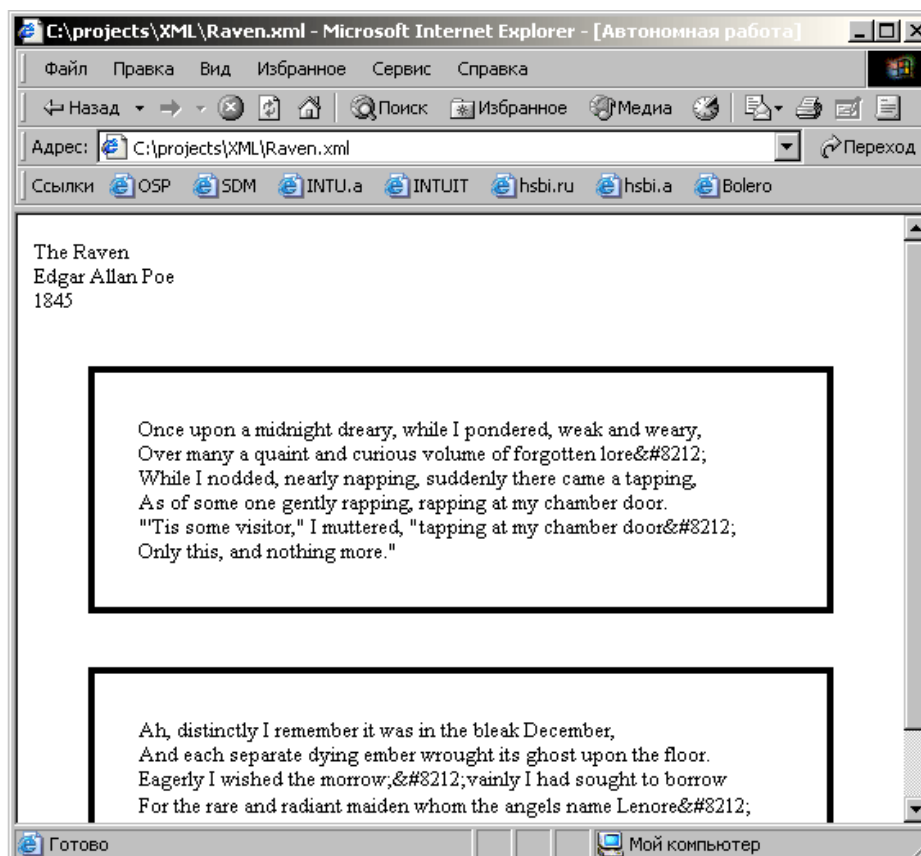


Рис. 7.32. Отображение в Internet Explorer 5 страницы, для которой установлены свойства просвета

⇒ *Примечание.* Как для содержимого элемента, для просвета отображается любой фон в виде сплошной заливки или рисунков, которые вы назначили для элемента.

7.11. УСТАНОВКА СВОЙСТВ РАЗМЕРОВ

Свойства `width` и `height` управляют размерами зоны содержимого элемента плюс областями, занимаемыми просветом и обрамлением.

Вы можете задавать свойствам `width` и `height` следующие типы значений:

– любое значение в размерных единицах, описанных ранее в 7.3.4 «Задание значений в размерных единицах». Например, следующее правило устанавливает для свойства `width` элемента STANZA значение в 3 дюйма, а для свойства `height` – значение в 2 дюйма:

```
STANZA
  {width:3in;
   height:2in}
```

– значение в процентах относительно ширины или высоты родительского элемента. Так, следующее правило задает для свойств `width` и `height` элемента STANZA значения, равные половине ширины и высоты родительского элемента:

```
STANZA
  {width:50%;
   height:50%}
```

– ключевое слово `auto`, которое устанавливается по умолчанию. При такой установке браузер подстраивает свойства `width` и `height` в соответствии с реальными размерами текста. Например, следующее правило задает для свойств `width` и `height` режим `auto` (это аналогично тому, что вы просто опустите установку свойств `width` и `height`):

```
STANZA
  {width:auto;
   height:auto}
```

Если вы сделаете ширину слишком маленькой, чтобы строки текста поместились внутри области содержимого, браузер будет переносить текст, пытаясь полностью разместить текст внутри отведенной области. Если же текст не помещается в зону содержимого по вертикали вследствие недостаточного значения, установленного для свойства `height`, браузер увеличит значение для `height`, чтобы разместить весь текст, как если бы вы установили для свойства `height` значение `auto`.

Так, если вы добавите следующее правило в таблицу стилей, содержащуюся в листинге 7.5, браузер отобразит XML-документ из листинга 7.6 (к которому присоединена таблица стилей), как показано на рис. 7.33:

```
STANZA
  {border-style:solid;
   width:2.5in;
   height:1in}
```

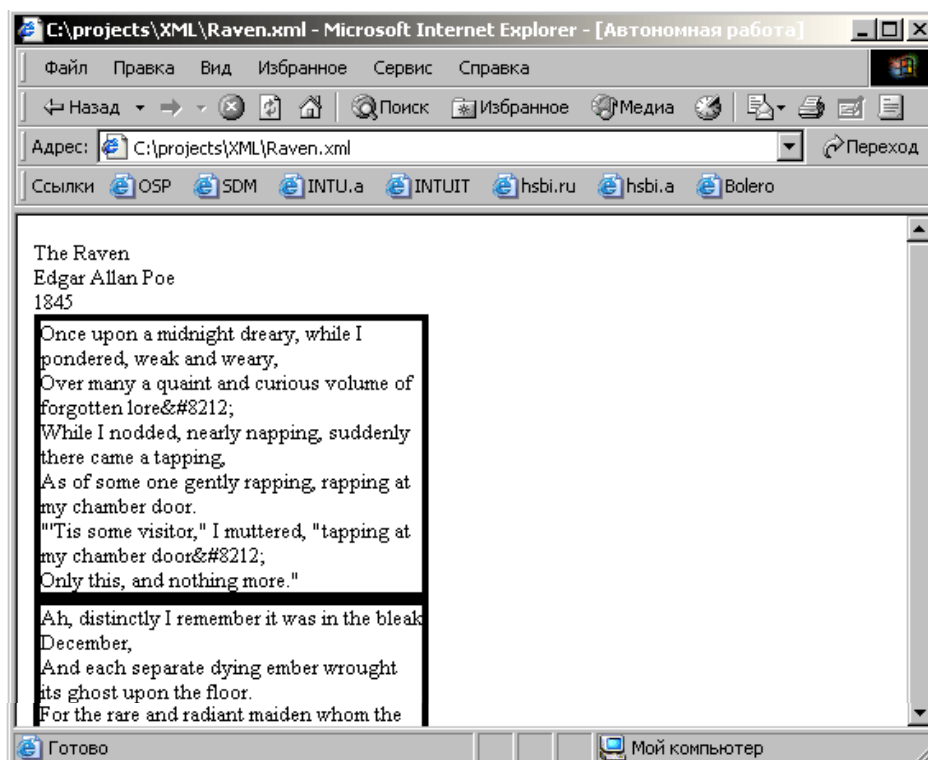


Рис. 7.33. Отображение в Internet Explorer 5 страницы, для которой установлены свойства размеров

Обратите внимание, что текст переносится с целью уместить его в отведенной ширине области в 2,5 дюйма, но высота области намного превосходит установленное значение в 1 дюйм, чтобы все содержимое текста было отображено.

7.12. УСТАНОВКА СВОЙСТВ ПОЗИЦИОНИРОВАНИЯ

Вы можете использовать свойства позиционирования `float` и `clear` для управления положением элемента `block` по отношению к следующему за ним тексту документа.

7.12.1. Установка свойства `float`

По умолчанию содержимое элемента `block` занимает всю ширину окна браузера, предшествующий текст документа размещается выше, а последующий текст документа — ниже содержимого элемента, подобно абзацу в обычном текстовом документе. Однако

у вас есть возможность воспользоваться свойством `float`, чтобы отобразить содержимое элемента `block` рядом (т. е. слева или справа) от следующего за элементом текста.

Вы можете устанавливать в качестве значений свойства `float` следующие три ключевых слова, представленных в табл. 7.15.

Таблица 7.15

Ключевые слова для свойства `float`

Ключевое слово	Описание
<code>left</code>	Отображает содержимое элемента слева от последующего текста документа
<code>right</code>	Отображает содержимое элемента справа от последующего текста документа
<code>none</code> (по умолчанию)	Отключает функцию обтекаемого размещения. Элемент при этом размещается как обычный элемент <code>block</code> , т. е. предшествующий текст документа – выше его, а последующий – ниже

В заданиях к данной главе вы научитесь использовать свойство `float` для создания поля примечаний, а также размещать обтекаемое изображение рядом с текстом элемента.

7.12.2. Установка свойства `clear`

По умолчанию обтекаемый элемент будет отображен слева или справа от последующего текста в документе. Однако вы можете отменить размещение элемента рядом с текстом, воспользовавшись свойством `clear`.

Свойству `clear` могут быть установлены следующие значения в виде ключевых слов, приведенных в табл. 7.16.

Таблица 7.16

Ключевые слова для свойства `clear`

Ключевое слово	Описание
<code>left</code>	Элемент будет отображен ниже (а не рядом) относительно предшествующего обтекаемого элемента, для которого установлено свойство <code>float:left</code>
<code>right</code>	Элемент будет отображен ниже (а не рядом) относительно предшествующего обтекаемого элемента, для которого установлено свойство <code>float:right</code>

Ключевое слово	Описание
both	Элемент будет отображен ниже (а не рядом) относительно предшествующего обтекаемого элемента, для которого установлено свойство float:left или float:right
none (по умолчанию)	Элемент будет отображен рядом с предшествующим обтекаемым элементом

Например, если вы добавите следующее правило в таблицу стилей из листинга 7.7, каждый элемент STANZA будет отображаться ниже предшествующего (обтекаемого) элемента IMAGE, но не рядом с ним (рис. 7.34):

```
STANZA
{clear:left}
```

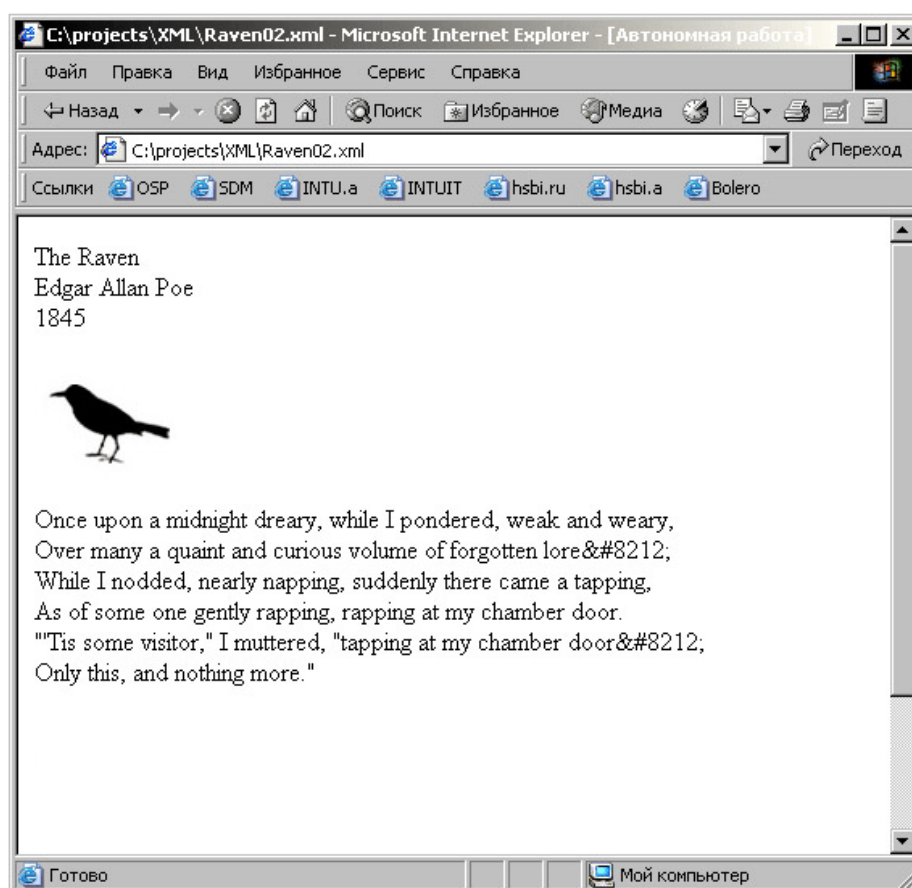


Рис. 7.34. Отображение в Internet Explorer 5 результата применения свойства clear (в тексте)

Листинг 7.7. Raven02.css

```
/* File Name: Raven02.css */
POEM
    {font-size:12pt}
POEM, TITLE, AUTHOR, DATE, IMAGE, STANZA, VERSE
    {display:block}
DATE, STANZA
    {margin-bottom:.25in}
IMAGE
    {background-image:url(Raven.bmp);
      background-repeat:no-repeat;
      background-position:center;
      width:89px;
      height:58px;
      float:left}
```

7.13. ВСТАВКА ЭЛЕМЕНТОВ HTML В XML-ДОКУМЕНТЫ И ИСПОЛЬЗОВАНИЕ ПРОСТРАНСТВА ИМЕН

Хотя вы можете использовать таблицы каскадных стилей для добавления базовых возможностей форматирования элементов XML в ваш документ, было бы хорошо иметь возможность добавлять стандартные HTML-элементы, такие как гиперссылки, изображения и формы, чтобы ваш документ выигрывал от применения встроенных в эти элементы функций. К счастью, когда вы отображаете ваш документ с помощью присоединенной таблицы стилей, вы можете вставить в ваш документ любой стандартный элемент HTML и заставить браузер отобразить этот элемент с использованием специальных зарезервированных для этой цели имен элементов.

Вам может показаться, что можно вставить HTML-элемент, просто присвоив XML-элементу такое же имя. Например, вставить HTML-элемент IMG, просто создав элемент IMG следующим образом:

```
<IMG SRC="Raven.bmp" />
```

Однако у браузера нет возможности узнать, что это HTML-элемент, а не обычный созданный вами XML-элемент. Чтобы подобный механизм мог работать, все имена HTML-элементов (а их много) должны были быть зарезервированы исключительно для вставки HTML-элементов. Такой подход, однако, противоречил бы духу XML, в соответствии с которым вам разрешается использовать для ваших элементов любые корректно заданные имена.

К счастью, вы можете воспользоваться соглашением XML, известным как пространство имен, которое позволяет различать конфликтующие имена. Два разных элемента могут иметь одно и то же имя, если они принадлежат разным пространствам имен.

Идентификатор пространства имен добавляется в начале имени элемента и отделяется от остальной части имени двоеточием (:), как в следующем примере:

```
my-namespace:MY-ELEMENT
```

Элемент с именем my-namespace:MY-ELEMENT и элемент с именем MY-ELEMENT могут существовать в одном и том же документе и считаются различными элементами, поскольку принадлежат отдельным пространствам имен: my-namespace:MY-ELEMENT принадлежит пространству имен my-namespace, а элемент MY-ELEMENT по умолчанию принадлежит к пространству имен документа.

Однако прежде чем вы сможете использовать пространство имен, вы должны соответствующим образом объявить его. Проще всего это сделать внутри начального тега элемента, для которого вы хотите использовать пространство имен. Например, вы можете объявить пространство имен my-namespace, как показано на рис. 7.35:

```
<namespace:MY-ELEMENT TEXT>  
  This is a text.  
</TEXT>
```

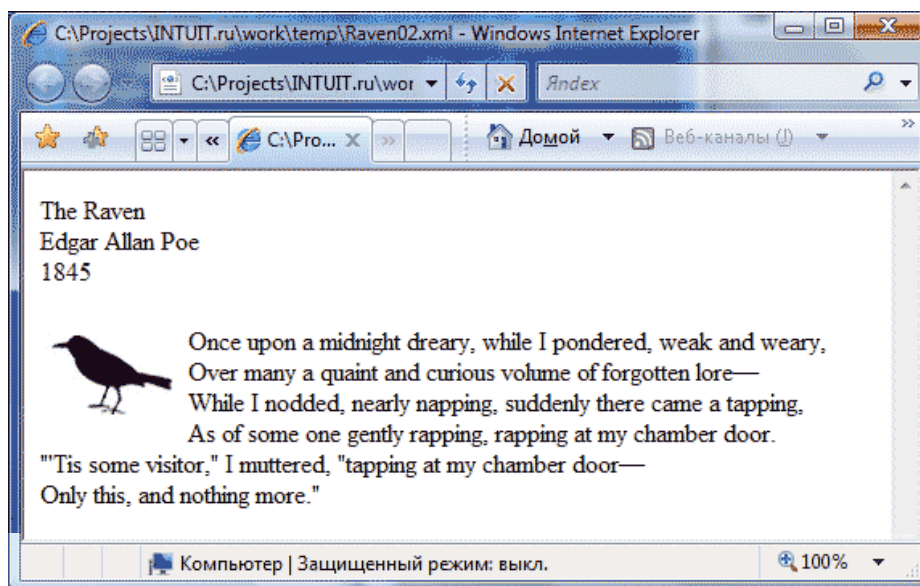


Рис. 7.35. Отображение в Internet Explorer 5 результата применения свойства clear (обтекание текстом)

Заметим, что при такой форме объявления пространства имен вы можете использовать пространство имен только внутри элемента, в котором вы его объявили, либо внутри любого из его дочерних элементов.

Если имя XML-элемента совпадает с именем стандартного HTML-элемента (например, IMG, A или HR) и если он принадлежит пространству имен html, Internet Explorer 5 интерпретирует его как HTML-элемент и поместит его на отображаемую страницу. Однако если элемент не принадлежит пространству имен html, Internet Explorer 5 интерпретирует его как обычный XML-элемент.

Пространство имен html является специальным, зарезервированным пространством имен, которое объявляется следующим образом:

```
xmlns:html='http://www.w3c.org/TR/REC-html40/'
```

Вот пример XML-элемента, который указывает Internet Explorer 5 вставить HTML-элемент IMG, для которого источником изображения является файл Raven.bmp:

```
<html:IMG xmlns:html='http://www.w3c.org/TR/REC-html40/'  
src='Raven.bmp' />
```

Это корректно сформированный пустой XML-элемент. Его имя включает указание пространства имен, он также имеет два атрибута. Первый атрибут объявляет пространство имен, в то время как второй атрибут является стандартным HTML-атрибутом, который вы можете включить в начальный тег HTML-элемента IMG.

Имейте в виду, что если вы вставляете HTML-элемент описанным выше способом, XML-документ должен иметь присоединенную таблицу стилей, и вы должны открыть его непосредственно в браузере.

Например, версия документа RAVEN, представленная в листинге 7.1 (см. на с. 127), иллюстрирует технику включения HTML в XML-документ. Обратите внимание, что к этому документу присоединена оригинальная версия таблицы стилей Raven.css, которая содержится в листинге 7.5 (см. на с. 178).

Документ включает три стандартных HTML-элемента:

1) он содержит изображение, представленное следующим XML-элементом:

```
<html:IMG xmlns:html='http://www.w3c.org/TR/REC-html40/'  
SRC='Raven.bmp' ALIGN='LEFT' />
```


Этот элемент помещает стандартный HTML-элемент IMG (изображение). Атрибут HTML ALIGN='LEFT' делает изображение объектовым и размещает его слева от последующего текста документа. Этот метод является альтернативой методу, с которым вы познакомитесь в заданиях к данной главе;

2) он делает имя автора (которое ранее содержалось в элементе AUTHOR) гиперссылкой с помощью следующего XML-элемента (вместо элемента AUTHOR):

```
<html:A xmlns:html='http://www.w3c.org/TR/REC-html40/'  
HREF='http://www.edgar.com'>  
Edgar Allan Poe  
</html:A>
```

Этот элемент вставляет стандартный элемент HTML A (якорь);

3) он вставляет две горизонтальные разделительные линии с использованием следующего XML-элемента:

```
<html:HR xmlns:xml='http://www.w3c.org/TR/REC-html40/' />
```

На рис. 7.36 показано, как Internet Explorer 5 отображает листинг 7.8.

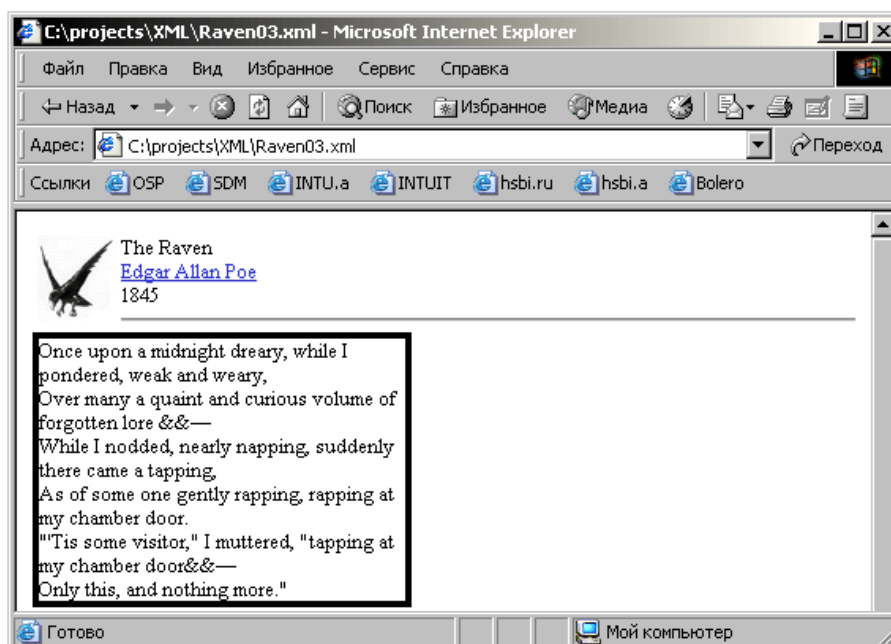


Рис. 7.36. Отображение в Internet Explorer 5 файла Raven03.xml

Листинг 7.8. Raven03.xml

```
<?xml version="1.0"?>  
<!-- File Name: Raven03.XML -->
```

```

<?xml-stylesheet type="text/css" href="Raven.css"?>
<POEM>
<html:IMG xmlns:html='http://www.w3c.org/TR/REC-html40/'
  SRC='Raven.bmp' ALIGN='LEFT' />
<TITLE>The Raven</TITLE>
<html:A xmlns:html='http://www.w3c.org/TR/REC-html40/'
  HREF='http://www.edgar.com'>
  Edgar Allan Poe
</html:A>
<DATE>1845</DATE>
<html:HR xmlns:html='http://www.w3c.org/TR/REC-html40/' />
<STANZA>
  <VERSE>Once upon a midnight dreary, while I
pondered, weak and weary,</VERSE>
  <VERSE>Over many a quaint and curious volume of
forgotten lore&#8212;</VERSE>
  <VERSE>While I nodded, nearly napping, suddenly
there came a tapping,</VERSE>
  <VERSE>As of some one gently rapping, rapping at my
chamber door.</VERSE>
  <VERSE>"'Tis some visitor," I muttered, "tapping at
my chamber door&#8212;</VERSE>
  <VERSE>Only this, and nothing more."</VERSE>
</STANZA>
<html:HR xmlns:html='http://www.w3c.org/TR/REC-html40/' />
<STANZA>
  <VERSE>Ah, distinctly I remember it was in the bleak
December,</VERSE>
  <VERSE>And each separate dying ember wrought its
ghost upon the floor.</VERSE>
  <VERSE>Eagerly I wished the morrow;&#8212;vainly
I had sought to borrow</VERSE>
  <VERSE>From my books surcease of sorrow&#8212;
sorrow for the lost
  Lenore&#8212;</VERSE>
  <VERSE>For the rare and radiant maiden whom the
angels name Lenore&#8212;</VERSE>
  <VERSE>Nameless here for evermore.</VERSE>
</STANZA>
</POEM>

```

7.14. СОЗДАНИЕ И ИСПОЛЬЗОВАНИЕ ПОЛНОЦЕННОЙ ТАБЛИЦЫ СТИЛЕЙ

В последующих упражнениях вы создадите XML-документ, содержащий первые четыре строфы поэмы Эдгара По «Ворон». Затем вы создадите таблицу каскадных стилей, которая отформатирует этот документ с использованием почти всех свойств, рассмотренных в этой главе. На рис. 7.37 показано, как поэма будет выглядеть в Internet Explorer 5.

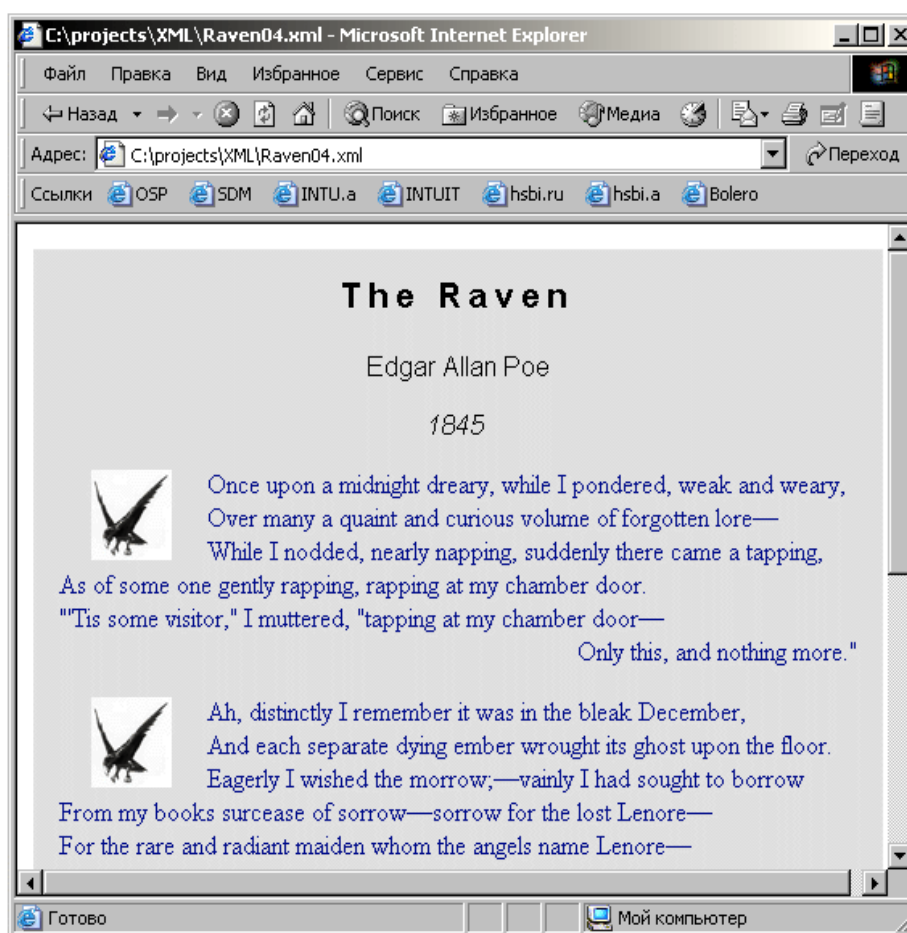


Рис. 7.37. Отображение в Internet Explorer 5 поэмы Э. По «Ворон»

? ЗАДАНИЯ

1. Создайте поле примечаний

В вашем текстовом редакторе откройте файл таблицы стилей Raven.css, приведенный в листинге 7.5 (см. на с. 178).

Модифицируйте таблицу стилей, чтобы она приняла вид, представленный в листинге 7.9.

```
Листинг 7.9. Raven01.css
/* File Name: Raven01.css */
POEM
    {font-size:12pt}
POEM, TITLE, AUTHOR, DATE, NOTE, STANZA, VERSE
    {display:block}
DATE
    {margin-bottom:.25in}
STANZA
    {margin-left:1in;
     margin-bottom:.25in}
NOTE
    {border-style:solid;
     border-width:1px;
     text-align:center;
     width:1in;
     height:1in;
     float:left}
```

К оригинальной таблице стилей добавлены следующие новшества. Для элементов STANZA установлено левое поле в 1 дюйм.

Элемент NOTE (который вы добавите в документ позднее) отформатирован как поле примечания, отображаемое в зоне левого поля первого элемента STANZA. Для этого:

- для него установлено сплошное обрамление с толщиной линий 1 пиксель;
- текст выровнен по центру;
- для свойств width и height установлено значение в 1 дюйм;
- установлен режим обтекания с размещением слева от последующего текста.

Воспользуйтесь командой Save As (Сохранить как) вашего текстового редактора, чтобы сохранить копию модифицированного документа под именем Raven01.css.

В вашем текстовом редакторе откройте документ Raven.xml, представленный в листинге 7.6 (см. на с. 178).

В файле Raven.xml отредактируйте инструкцию по обработке xml-stylesheet в начале файла, чтобы он указывал на новую таблицу стилей, которую вы только что создали – Raven01.css – следующим образом:

```
<?xml-stylesheet type="text/css" href="Raven01.css"?>
```

В документе Raven.xml добавьте следующий новый элемент непосредственно над элементом STANZA:

```
<NOTE>This is a floating margin note.</NOTE>
```

Поскольку в таблице стилей для элемента NOTE вы установили свойство float:left, он будет располагаться слева от последующего текста документа, т. е. слева от первого элемента STANZA.

Выберите команду Save As (Сохранить как) вашего текстового редактора, чтобы сохранить копию модифицированного документа под именем Raven01.xml.

Весь документ представлен в листинге 7.10.

Листинг 7.10. Raven01.xml (html, txt)

```
<?xml version="1.0"?>
<!-- File Name: Raven01.xml -->
<?xml-stylesheet type="text/css" href="Raven01.css"?>
<POEM>
<TITLE>The Raven</TITLE>
<AUTHOR>Edgar Allan Poe</AUTHOR>
<DATE>1845</DATE>
<NOTE>This is a floating margin note.</NOTE>
<STANZA>
  <VERSE>Once upon a midnight dreary, while I
pondered, weak and weary,</VERSE>
  <VERSE>Over many a quaint and curious volume of
forgotten lore;&#8212;</VERSE>
  <VERSE>While I nodded, nearly napping, suddenly
there came a tapping,</VERSE>
  <VERSE>As of some one gently rapping, rapping at my
chamber door.</VERSE>
  <VERSE>"'Tis some visitor," I muttered, "tapping at
my chamber door&#8212;</VERSE>
  <VERSE>Only this, and nothing more."</VERSE>
</STANZA>
<STANZA>
  <VERSE>Ah, distinctly I remember it was in the bleak
December,</VERSE>
  <VERSE>And each separate dying ember wrought its
ghost upon the floor.</VERSE>
  <VERSE>Eagerly I wished the morrow;&#8212;vainly I
had sought to borrow</VERSE>
```

```
<VERSE>From my books surcease of sorrow&#8212;sorrow  
for the lost Lenore&#8212;</VERSE>
```

```
<VERSE>For the rare and radiant maiden whom the  
angels name Lenore&#8212;</VERSE>
```

```
<VERSE>Nameless here for evermore.</VERSE>  
</STANZA>  
</POEM>
```

Откройте файл Raven01.xml в Internet Explorer 5. Он будет иметь вид, как показано на рис. 7.38.

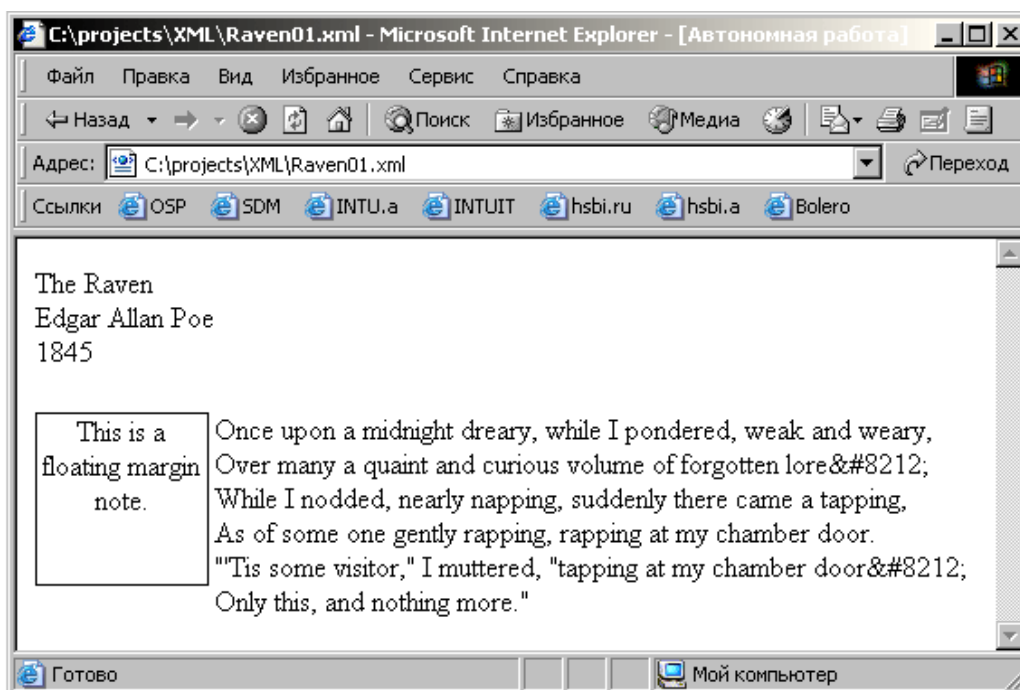


Рис. 7.38. Отображение в Internet Explorer 5 файла Raven01.xml

2. Отобразите обтекаемое текстом изображение

В вашем текстовом редакторе откройте таблицу стилей Raven.css, представленную в листинге 7.5 (см. на с. 178).

Модифицируйте таблицу стилей в соответствии с листингом 7.7 (см. на с. 190).

Главным дополнением является введение правила для элемента IMAGE:

```
IMAGE  
{background-image:url(Raven.bmp);  
background-repeat:no-repeat;  
background-position:center;
```

```
width:89px;  
height:58px;  
float:left}
```

Элемент `IMAGE` является пустым элементом (его вы позднее добавите в XML-документ), созданным для отображения обтекаемого текстом рисунка. Элемент не содержит текста, но ему назначен фоновый рисунок (с помощью трех свойств, устанавливаемых в правиле), который будет отображен вместо текста.

Свойствам `width` и `height` элемента присвоены точные значения ширины и высоты изображения. Поскольку файл рисунка является растровым (точечным), важно указать размер в пикселях, чтобы изображение могло быть полностью отображено на любом мониторе в любом графическом режиме. Заметим, что если вы не присвоите значений свойствам `width` и `height` элемента, размер его будет нулевым, поскольку он не содержит текста, и поэтому будет скрыт.

Наконец, вы установили для свойства `float` значение `left`, чтобы изображение располагалось слева от обтекающего его текста документа.

Воспользуйтесь командой `Save As` (Сохранить как) вашего текстового редактора, чтобы сохранить копию модифицированного документа под именем `Raven02.css`. (Данный документ отображен в листинге 7.7.)

В вашем текстовом редакторе откройте документ `Raven.xml`, приведенный в листинге 7.6 (см. на с. 178).

В документе `Raven.xml` отредактируйте инструкцию по обработке `xml-styleSheet` в начале файла, чтобы она указывала на новую таблицу стилей, которую вы только что создали – `Raven02.css` – следующим образом:

```
<?xml-styleSheet type="text/css" href="Raven02.css"?>
```

В документ `Raven.xml` добавьте следующий пустой элемент `IMAGE` непосредственно перед каждым элементом `STANZA`:

```
<IMAGE/>
```

Поскольку вы назначили элементам `IMAGE` свойство `float:left`, они будут обтекаться текстом и располагаться слева от элемента `STANZA` (который содержит последующий текст документа).

Выберите команду Save As (Сохранить как) текстового редактора, чтобы сохранить копию модифицированного документа под именем Raven02.xml.

Полный документ представлен в листинге 7.11.

Листинг 7.11. Raven02.xml

```
<?xml version="1.0"?>
<!-- File Name: Raven02.xml -->
<?xml-stylesheet type="text/css" href="Raven02.css"?>
<POEM>
<TITLE>The Raven</TITLE>
<AUTHOR>Edgar Allan Poe</AUTHOR>
<DATE>1845</DATE>
<IMAGE/>
<STANZA>
    <VERSE>Once upon a midnight dreary, while I
pondered, weak and weary,</VERSE>
    <VERSE>Over many a quaint and curious volume of
forgotten lore&#8212;</VERSE>
    <VERSE>While I nodded, nearly napping, suddenly
there came a tapping,</VERSE>
    <VERSE>As of some one gently rapping, rapping at my
chamber door.</VERSE>
    <VERSE>"'Tis some visitor," I muttered, "tapping at
my chamber door&#8212;</VERSE>
    <VERSE>Only this, and nothing more."</VERSE>
</STANZA>
<IMAGE/>
<STANZA>
    <VERSE>Ah, distinctly I remember it was in the
bleak December,</VERSE>
    <VERSE>And each separate dying ember wrought its
ghost upon the floor.</VERSE>
    <VERSE>Eagerly I wished the morrow;&#8212;vainly I
had sought to borrow</VERSE>
    <VERSE>From my books surcease of sorrow&#8212;sorrow
for the lost Lenore&#8212;</VERSE>
    <VERSE>For the rare and radiant maiden whom the
angels name Lenore&#8212;</VERSE>
    <VERSE>Nameless here for evermore.</VERSE>
</STANZA>
</POEM>
```


Откройте файл Raven02.xml в Internet Explorer 5. Он будет отображен, как показано на рис. 7.39.

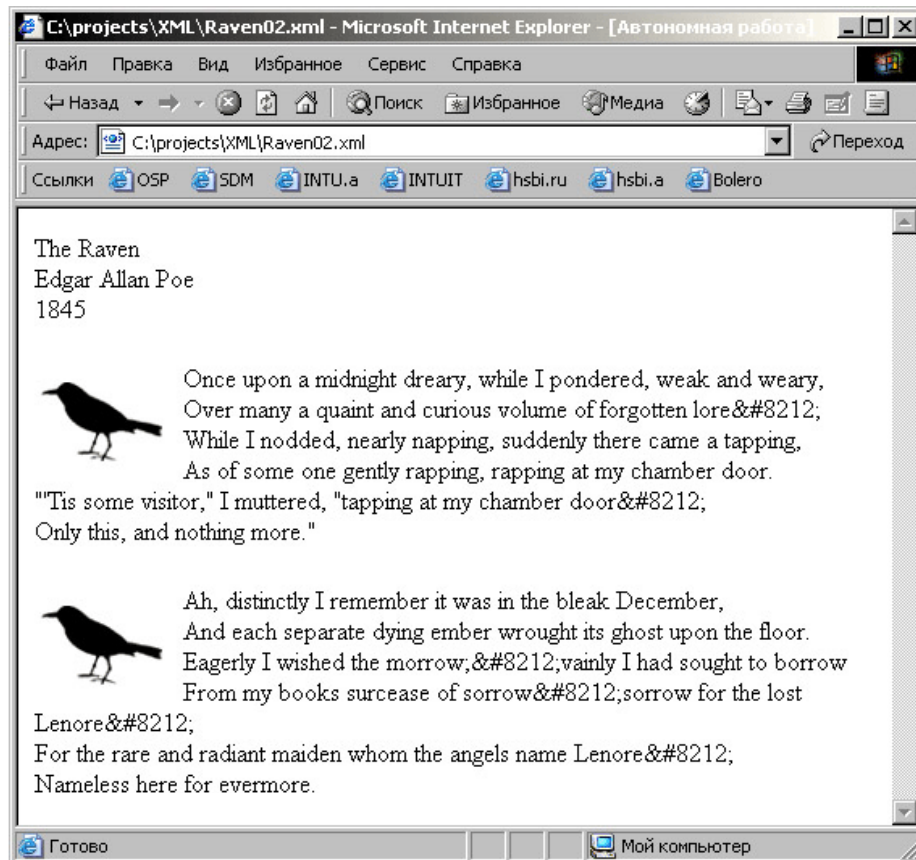


Рис. 7.39. Отображение в Internet Explorer 5 файла Raven02.xml

3. Создайте документ

Откройте новый, пустой текстовый файл в вашем текстовом редакторе и введите XML-документ, приведенный в листинге 7.12.

Листинг 7.12. Raven03.xml

```
<?xml version="1.0"?>
<!-- File Name: Raven03.xml -->
<?xml-stylesheet type="text/css" href="Raven04.css"?>
<POEM>
<TITLE>The Raven</TITLE>
<AUTHOR>
  Edgar Allan Poe
  <AUTHOR-BIO>Edgar Allan Poe was an American writer
who lived from 1809 to 1849.</AUTHOR-BIO>
</AUTHOR>
<DATE>1845</DATE>
```

<IMAGE/>
 <STANZA>
 <VERSE>Once upon a midnight dreary, while I
 pondered, weak and weary,</VERSE>
 <VERSE>Over many a quaint and curious volume of
 forgotten lore—</VERSE>
 <VERSE>While I nodded, nearly napping, suddenly
 there came a tapping,</VERSE>
 <VERSE>As of some one gently rapping, rapping at
 my chamber door.</VERSE>
 <VERSE>"'Tis some visitor," I muttered, "tapping
 at my chamber door—</VERSE>
 <LASTVERSE>Only this, and nothing more."</LASTVERSE>
 </STANZA>
 <IMAGE/>
 <STANZA>
 <VERSE>Ah, distinctly I remember it was in the
 bleak December,</VERSE>
 <VERSE>And each separate dying ember wrought its
 ghost upon the floor.</VERSE>
 <VERSE>Eagerly I wished the morrow;—vainly I
 had sought to borrow</VERSE>
 <VERSE>From my books surcease of
 sorrow—sorrow for the lost Lenore—</VERSE>
 <VERSE>For the rare and radiant maiden whom the
 angels name Lenore—</VERSE>
 <LASTVERSE>Nameless here for evermore.</LASTVERSE>
 </STANZA>
 <IMAGE/>
 <STANZA>
 <VERSE>And the silken sad uncertain rustling of
 each purple curtain</VERSE>
 <VERSE>Thrilled me—filled me with fantastic
 terrors never felt before;</VERSE>
 <VERSE>So that now, to still the beating of my
 heart, I stood repeating:</VERSE>
 <VERSE>"'Tis some visitor entreating entrance at
 my chamber door—</VERSE>
 <VERSE>Some late visitor entreating entrance at my
 chamber door;</VERSE>
 <LASTVERSE>This it is, and nothing more."</LASTVERSE>
 </STANZA>
 <IMAGE/>
 <STANZA>

```

    <VERSE>Presently my soul grew stronger; hesitating
then no longer,</VERSE>
    <VERSE>"Sir," said I, "or Madam, truly your forgiveness
I implore;</VERSE>
    <VERSE>But the fact is I was napping, and so
gently you came rapping,</VERSE>
    <VERSE>And so faintly you came tapping, tapping at
my chamber door,</VERSE>
    <VERSE>That I scarce was sure I heard you"&#8212;here
I opened wide the door;&#8212;</VERSE>
    <LASTVERSE>Darkness there, and nothing more.</LASTVERSE>
</STANZA>
</POEM>

```

Обратите внимание на следующие важные особенности документа Raven03.xml:

- 1) к нему присоединена таблица каскадных стилей Raven04.css, которую вы создадите в следующем упражнении;
- 2) перед каждым элементом STANZA расположен пустой элемент IMAGE. Вы используете элемент IMAGE для отображения рисунка ворона в начале каждой строфы;
- 3) последняя строка в каждой строфе помещена в специальный элемент с именем LASTVERSE. Это дает вам возможность форматировать последнюю строку отличным образом от остальных строк. (Она выровнена не по левому краю, а по правому.)

Воспользуйтесь командой Save (Сохранить) вашего текстового редактора, чтобы сохранить документ на вашем жестком диске под именем Raven03.xml.

4. Создайте таблицу стилей

Откройте новый, пустой файл в вашем текстовом редакторе и введите таблицу каскадных стилей, представленную в листинге 7.13.

Листинг 7.13. Raven04.css

```

/* File Name: Raven04.css */
POEM
{font-size:12pt;
width:5.5in;
padding:1em;
border-width:1px;
background-color:rgb(225,225,225)}
POEM, TITLE, AUTHOR, DATE, STANZA
{display:block;
margin-bottom:1em}

```

```

AUTHOR-BIO
    {display:none}
TITLE, AUTHOR, DATE
    {font-family:Arial,sans-serif;
    text-align:center}
DATE
    {font-style:italic}
TITLE
    {font-size:16pt;
    font-weight:bold;
    letter-spacing:.25em}
IMAGE
    {background-image:url(RavShade.bmp);
    background-repeat:no-repeat;
    background-position:center;
    width:89px;
    height:58px;
    float:left}
STANZA
    {color:navy;
    line-height:1.25em}
VERSE
    {display:block}
LASTVERSE
    {display:block;
    text-align:right}

```

Относительно этой таблицы стилей следует иметь в виду:

- таблица стилей демонстрирует почти все свойства, изученные в этой главе;

- все использованные в таблице стилей приемы были рассмотрены в предыдущих подглавах данной главы;

- файл рисунка (RavShade.bmp), отображенный с использованием элементов IMAGE, представляет собой тот же рисунок, который вы отображали в предшествующих версиях документа Raven.xml, за исключением того, что он имеет теневой фон, соответствующий цвету фона элемента POEM;

- таблица стилей скрывает содержимое элемента AUTHOR-BIO путем присвоения значения none его свойству display.

Выберите команду Save (Сохранить), чтобы сохранить документ на вашем жестком диске под именем Raven04.css.

Отобразите документ, открыв файл Raven04.xml непосредственно в Internet Explorer 5 (см. рис. 7.37 на с. 195).

Глава 8. ОТОБРАЖЕНИЕ XML-ДОКУМЕНТА. СВЯЗЬ ДАННЫХ

8.1. ОСНОВНЫЕ ШАГИ

Существует два основных этапа при связывании данных:

1) установка связи XML-документа с HTML-страницей, на которой вы хотите отобразить данные XML. Этот шаг обычно реализуется включением HTML-элемента с именем XML в HTML-страницу. Например, следующий элемент на HTML-странице связывает XML-документ Book.xml со страницей:

```
<XML ID="dsoBook" SRC="Book.xml"></XML>
```

2) сцепление HTML-элементов с XML-элементами. Когда вы сцепляете HTML-элементы с XML-элементом, HTML-элемент автоматически отображает содержимое XML-элемента. Например, следующий элемент SPAN на HTML-странице сцеплен с элементом AUTHOR связанного XML-документа:

```
<SPAN DATASRC="#dsoBook" DATAFLD="AUTHOR"></SPAN>
```

В результате HTML-элемент SPAN отображает содержимое XML-элемента AUTHOR.

Базовая технология связывания данных в действительности столь же проста, как и в этом примере, хотя в дальнейшем вы познакомитесь с различными вариациями и способами использования данной технологии.

Чтобы отобразить XML-документ на HTML-странице, вы должны установить его связь со страницей. Самый простой путь сделать это в Microsoft Internet Explorer 7 – включить в страницу HTML-элемент с именем XML, так называемый фрагмент данных. Вы можете использовать одну из двух различных форм записи для фрагмента данных.

В первой форме весь текст XML-документа помещается между начальным и конечным тегами XML. Вот пример фрагмента данных на HTML-странице:

```
<HTML>
<HEAD>
  <TITLE>Book Description</TITLE>
```

```

</HEAD>
<BODY>
  <XML ID="dsoBook">
    <?xml version="1.0"?>
    <BOOK>
      <TITLE>The Adventures of Huckleberry Finn</TITLE>
      <AUTHOR>Mark Twain</AUTHOR>
      <BINDING>mass market paperback</BINDING>
      <PAGES>298</PAGES>
      <PRICE>$5.49</PRICE>
    </BOOK>
  </XML>
</BODY>
</HTML>

```

Во второй форме записи HTML-элемент с именем XML остается пустым и содержит только URL XML-документа. Ниже приведен пример фрагмента данных на HTML-странице:

```

<HTML>
<HEAD>
  <TITLE>Book Description</TITLE>
</HEAD>
<BODY>
  <XML ID="dsoBook" SRC="Book.xml"></XML>
  <!-- Другие элементы HTML... -->
</BODY>
</HTML>

```

В предыдущем примере текст XML-документа должен содержаться в отдельном файле Book.xml:


```

<?xml version="1.0"?>
<!-- Имя файла: Book.xml -->
<BOOK>
  <TITLE>The Adventures of Huckleberry Finn</TITLE>
  <AUTHOR>Mark Twain</AUTHOR>
  <BINDING>mass market paperback</BINDING>
  <PAGES>298</PAGES>
  <PRICE>$5.49</PRICE>
</BOOK>

```

Вторая форма более соответствует основам философии XML, согласно которой собственно данные (XML-документ) хранятся отдельно от информации по их форматированию и обработке (таблицы

стилей или, в данном случае, HTML-страницы). Вторая форма облегчает работу с XML-документом, особенно если один документ отображается на нескольких различных HTML-страницах. В рассматриваемых в этом курсе примерах вы будете иметь дело только со второй формой записи фрагмента данных.

 *Примечание.* Имейте в виду, что элемент с именем XML, используемый для создания фрагмента данных, не является собственно XML-элементом. Это просто HTML-элемент, который содержит XML-элементы. Следовательно, использование синтаксиса XML для пустого элемента недопустимо:

```
<XML ID="dsoBook" SRC="Book.xml"/><!-- недопустимо -->
```

Вы должны присвоить атрибуту ID фрагмента данных уникальный идентификатор, который используете для доступа к XML-документу с HTML-страницы. (В предыдущем примере в качестве значения для ID выступает «dsoBook».)

При второй форме записи фрагмента данных вы присваиваете атрибуту SRC URL файла, содержащего данные XML. Вы можете использовать полный URL, как в следующем примере:

```
<XML ID="dsoBook" SRC=http://www.my_domain.com/Book.xml>
</XML>
```

Чаще, однако, вы используете частичный URL, который задает местонахождение относительно местонахождения HTML-страницы, содержащей фрагмент данных. Например, атрибут SRC в следующем фрагменте данных указывает, что файл Book.xml находится в той же папке, что и HTML-страница:

```
<XML ID="dsoBook" SRC="Book.xml"></XML>
```

Относительные URL более распространены, потому что XML-документ обычно содержится в той же папке, что и HTML-страница, либо в одной из вложенных папок.

8.2. КАК ХРАНЯТСЯ ДАННЫЕ XML

Когда Internet Explorer 7 открывает HTML-страницу, его встроенный XML-процессор синтаксически анализирует XML-документ. Internet Explorer 7 также создает программный объект, который

носит название объект исходных данных (Data Source Object – DSO), хранит данные XML и обеспечивает доступ к этим данным. DSO хранит данные XML как набор записей, т. е. множество записей и их полей. Например, если вы включите документ Inventory.xml (см. листинг 8.1 на с. 210–211) в страницу как фрагмент данных, DSO будет хранить каждый элемент BOOK как запись, а каждый дочерний элемент внутри BOOK (TITLE, AUTHOR и т. д.) как поле.

Когда вы сцепляете HTML-элемент с XML-элементом, DSO автоматически предоставляет значение XML-элемента и управляет всеми его свойствами. DSO также позволяет вам напрямую осуществлять доступ и манипулирование имеющимся набором записей посредством ряда методов, свойств и событий. Методы представляют собой функции, которые вы можете вызывать со страницы для доступа или модификации набора записей. (Например, вы можете использовать методы для перемещения между записями.) Свойства представляют собой установленные на данный момент параметры, которые вы можете считывать и в ряде случаев изменять со страницы. (Например, вы можете считать свойство, которое сообщает вам, достигли ли вы последней записи.) События представляют собой определенные смены состояний (например, изменение значений записи), которыми вы можете управлять посредством функции сценария, который вы создаете для страницы.

На странице идентификатор, который вы присваиваете атрибуту ID во фрагменте данных, представляет DSO. (В предыдущей подглаве в рассмотренном примере таким идентификатором является dsoBook.)

8.3. ПРОВЕРКА НА НАЛИЧИЕ ОШИБОК XML

Когда вы открываете XML-документ (автономный или с таблицей стилей) непосредственно в Internet Explorer 7, браузер проверяет, является ли документ корректно сформированным. Если он обнаруживает ошибки, то приостанавливает отображение документа и выводит сообщение о фатальной ошибке, которое помогает вам выявить ошибку и устранить ее.

Если вы открываете XML-документ через фрагмент данных на HTML-странице, Internet Explorer 5 проверяет, является ли документ корректно сформированным, а также – если документ включает объявление типа документа – является ли он валидным. Однако в том

случае, если документ содержит ошибку, Internet Explorer 5 просто не будет отображать данные XML, не выводя сообщение об ошибке.

Чтобы увидеть описание какой-либо ошибки, имеющейся в связанном XML-документе, вы можете протестировать документ с использованием сценариев проверки на корректность и валидность, представленных в пункте «Проверка валидности XML-документа» (см. на с. 291) в главе 9.

Вы можете осуществлять сцепление HTML-элементов с XML-элементами двумя основными способами:

1) табличное сцепление, что означает сцепление HTML-элемента TABLE с данными XML, так что в таблице автоматически отображается весь набор записей, принадлежащих XML-документу;

2) сцепление по отдельным записям, что означает сцепление нетабличных элементов HTML с XML-элементами таким образом, что за один раз отображается только одна запись.

8.4. ИСПОЛЬЗОВАНИЕ ТАБЛИЧНОГО СЦЕПЛЕНИЯ ДАННЫХ

Самый простой способ отобразить XML-документ, который состоит из группы записей (такой как Inventory.xml, представленный в листинге 8.1), – это сцепить HTML-элемент TABLE с данными XML таким образом, чтобы в таблице автоматически отображались сразу все записи (или одна страница записей за раз, если вы установили режим постраничного отображения). При таком подходе Internet Explorer 7 берет на себя большую часть работы; вам не нужно писать сценарии или вызывать методы (функции).

Вы можете использовать одну таблицу HTML для отображения XML-документа, структурированного как набор записей, либо вы можете использовать вложенные HTML-таблицы для отображения XML-документа, содержащего иерархический набор записей (более сложную структуру записей).

8.4.1. Использование одной HTML-таблицы для отображения простого набора записей

Вы можете использовать один HTML-элемент TABLE для отображения XML-документа, в котором данные организованы в виде простого набора записей, т. е. XML-документа, составленного следующим образом:

– корневой элемент содержит множество элементов типа запись (record) (в этой главе подобные элементы иногда называются просто записями);

– каждый элемент типа запись содержит одинаковый набор элементов типа поле (field) (в этой главе подобные элементы иногда называются просто полями);

– каждый элемент типа поле содержит только символьные данные.

Примером такого типа XML-документов является документ `Inventory.xml`, который вы использовали в предыдущих главах. Он представлен в листинге 8.1. В этом документе корневой элемент (INVENTORY) содержит набор из восьми элементов-записей (элементы BOOK), и каждый из элементов-записей имеет одинаковый набор элементов-полей, которые содержат только символьные данные (TITLE, AUTHOR, BINDING, PAGES, PRICE).

Листинг 8.1. Inventory.xml

```
<?xml version="1.0"?>
<!-- Имя файла: Inventory.xml -->
<INVENTORY>
  <BOOK>
    <TITLE>The Adventures of Huckleberry Finn</TITLE>
    <AUTHOR>Mark Twain</AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>298</PAGES>
    <PRICE>$5.49</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>Leaves of Grass</TITLE>
    <AUTHOR>Walt Whitman</AUTHOR>
    <BINDING>hardcover</BINDING>
    <PAGES>462</PAGES>
    <PRICE>$7.75</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>The Legend of Sleepy Hollow</TITLE>
    <AUTHOR>Washington Irving</AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>98</PAGES>
    <PRICE>$2.95</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>The Marble Faun</TITLE>
    <AUTHOR>Nathaniel Hawthorne</AUTHOR>
    <BINDING>trade paperback</BINDING>
```

```

        <PAGES>473</PAGES>
        <PRICE>$10.95</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>Moby-Dick</TITLE>
        <AUTHOR>Herman Melville</AUTHOR>
        <BINDING>hardcover</BINDING>
        <PAGES>724</PAGES>
        <PRICE>$9.95</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>The Portrait of a Lady</TITLE>
        <AUTHOR>Henry James</AUTHOR>
        <BINDING>mass market paperback</BINDING>
        <PAGES>256</PAGES>
        <PRICE>$4.95</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>The Scarlet Letter</TITLE>
        <AUTHOR>Nathaniel Hawthorne</AUTHOR>
        <BINDING>trade paperback</BINDING>
        <PAGES>253</PAGES>
        <PRICE>$4.25</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>The Turn of the Screw</TITLE>
        <AUTHOR>Henry James</AUTHOR>
        <BINDING>trade paperback</BINDING>
        <PAGES>384</PAGES>
        <PRICE>$3.35</PRICE>
    </BOOK>
</INVENTORY>

```

Когда вы связываете таблицу с XML-документом, данные, принадлежащие каждому из элементов записей, отображаются в отдельной строке таблицы, а каждый из дочерних элементов полей – в отдельном столбце.

В качестве примера возьмем HTML-страницу из листинга 8.2, которая содержит таблицу, сцепленную с данными документа Inventory.xml из листинга 8.1.

Листинг 8.2. Inventory Table.htm

```

<!-- Имя файла: Inventory Table.htm -->
<HTML>
<HEAD>

```

```

        <TITLE>Book Inventory</TITLE>
    </HEAD>
    <BODY>
        <XML ID="dsoInventory" SRC="Inventory.xml"></XML>
        <H2>Book Inventory</H2>
        <TABLE DATASRC="#dsoInventory" BORDER="1"
CELLPADDING="5">
            <THEAD>
                <TH>Title</TH>
                <TH>Author</TH>
                <TH>Binding</TH>
                <TH>Pages</TH>
                <TH>Price</TH>
            </THEAD>
            <TR ALIGN="center">
                <TD><SPAN DATAFLD="TITLE"
STYLE="font-style:italic"></SPAN></TD>
                <TD><SPAN DATAFLD="AUTHOR"></SPAN></TD>
                <TD><SPAN DATAFLD="BINDING"></SPAN></TD>
                <TD><SPAN DATAFLD="PAGES"></SPAN></TD>
                <TD><SPAN DATAFLD="PRICE"></SPAN></TD>
            </TR>
        </TABLE>
    </BODY>
</HTML>

```

XML-документ из листинга 8.1 связан с HTML-страницей из листинга 8.2 посредством фрагмента данных на этой странице, имеющего ID dsoInventory:

```
<XML ID="dsoInventory" SRC="Inventory.xml"></XML>
```

Элемент TABLE страницы сцеплен со всем XML-документом путем присвоения атрибуту DATASRC элемента идентификатора (ID) фрагмента данных, предваренного символом #:

```
<TABLE DATASRC="#dsoInventory" BORDER="1" CELLPADDING="5">
```

Таблица определена со стандартным заголовком (элемент THEAD) и с одной строкой (элемент TR). Каждая ячейка в этой строке (т. е. каждый элемент TD) состоит из элемента SPAN, который сцеплен с одним из полей XML-документа таким образом, что этот элемент отображает содержимое поля. Например, первая ячейка содержит элемент SPAN, сцепленный с полем TITLE:

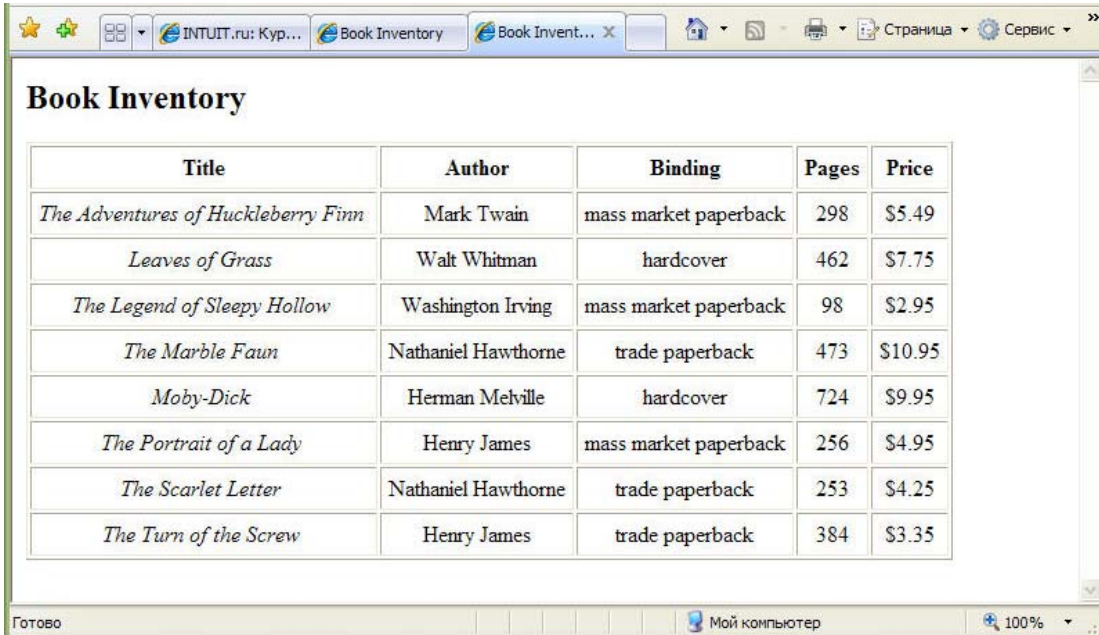
```

<TD><SPAN DATAFLD="TITLE"
STYLE="font-style:italic"></SPAN></TD>

```

Элемент SPAN связывается с полем XML путем присвоения имени поля (в данном примере TITLE) атрибуту DATAFLD элемента.

Вот как работает связывание данных. Даже если в элементе TABLE определена только одна строка, когда браузер отображает таблицу, он повторяет строковый элемент для каждой записи в XML-документе (рис. 8.1). Иными словами, в первой строке, следующей за заголовком, отображены поля (TITLE, AUTHOR и т. д.), принадлежащие первой записи (элемент BOOK для книги *The Adventures of Huckleberry Finn*). В следующей строке отображены поля для второй записи (элемент BOOK для книги *Leaves of Grass*) и т. д.



Title	Author	Binding	Pages	Price
<i>The Adventures of Huckleberry Finn</i>	Mark Twain	mass market paperback	298	\$5.49
<i>Leaves of Grass</i>	Walt Whitman	hardcover	462	\$7.75
<i>The Legend of Sleepy Hollow</i>	Washington Irving	mass market paperback	98	\$2.95
<i>The Marble Faun</i>	Nathaniel Hawthorne	trade paperback	473	\$10.95
<i>Moby-Dick</i>	Herman Melville	hardcover	724	\$9.95
<i>The Portrait of a Lady</i>	Henry James	mass market paperback	256	\$4.95
<i>The Scarlet Letter</i>	Nathaniel Hawthorne	trade paperback	253	\$4.25
<i>The Turn of the Screw</i>	Henry James	trade paperback	384	\$3.35

Рис. 8.1. Отображение в Internet Explorer 5 связи данных элементов TD

У вас может возникнуть вопрос, почему ячейки (элементы TD) не сцеплены непосредственно с полями XML. Ответ заключается в том, что элемент TD не является связываемым HTML-элементом. Следовательно, вы должны включить внутрь каждого элемента TD связываемый элемент (обычно SPAN).

8.4.2. Использование постраничного отображения

Если XML-документ содержит много записей, вы можете использовать постраничный вывод группы записей за один раз вместо отображения всех записей одновременно в огромной таблице.

Для активизации постраничного отображения в обычной связанной таблице выполните следующие действия.

Установите для атрибута DATAPAGESIZE сцепленного элемента TABLE значение, равное максимальному числу записей, которые вы хотите отобразить за раз. Каждая страница записей будет содержать заданное вами число записей. Например, следующий начальный тег для элемента TABLE присваивает число «5» атрибуту DATAPAGESIZE, в результате чего в таблице будет отображено пять записей за раз:

```
<TABLE DATASRC="#dsoInventory" DATAPAGESIZE="5">
```

Присвойте уникальный идентификатор атрибуту ID элемента TABLE, как для следующего начального тега:

```
<TABLE ID="InventoryTable" DATASRC="#dsoInventory"
DATAPAGESIZE="5">
```

Чтобы перемещаться между записями, вызывайте методы элемента TABLE, представленные в табл. 8.1. Для приведенных в последнем столбце примеров предполагается, что таблица имеет идентификатор InventoryTable.

Таблица 8.1

Методы элемента TABLE

Метод элемента TABLE	Эффект	Пример вызова
firstPage	Отображает первую страницу записей	InventoryTable.firstPage()
previousPage	Отображает предыдущую страницу записей	InventoryTable.previousPage()
nextPage	Отображает следующую страницу записей	InventoryTable.nextPage()
lastPage	Отображает последнюю страницу записей	InventoryTable.lastPage()

Если в текущий момент отображена первая страница, вызов метода previousPage игнорируется, а если отображена последняя страница, то игнорируется вызов nextPage.

Вы можете вызвать любой из этих методов из написанного вами сценария. Однако наиболее простой способ обращения к одному из методов заключается в присвоении метода атрибуту ONCLICK HTML-элемента BUTTON, как в следующем примере:

```
<BUTTON ONCLICK="InventoryTable.nextPage()">Next Page</BUTTON>
```

Этот элемент отображает кнопку. Когда пользователь щелкает на кнопке, вызывается метод, присвоенный атрибуту ONCLICK, а именно, `InventoryTable.nextPage`.

Техника использования постраничного вывода демонстрируется в листингах 8.3 и 8.4. Листинг 8.3 представляет собой расширенную версию документа `Inventory.xml` из листинга 8.1. Листинг 8.4 (см. на с. 218–219) представляет собой HTML-страницу, которая отображает этот XML-документ в таблице, атрибуту `DATAPAGESIZE` которой присвоено значение «5».

В верхней части страницы (рис. 8.2) имеется четыре элемента `BUTTON`, каждый из которых выполняет действие в соответствии с методами постраничного вывода таблицы. Когда вы впервые открываете HTML-страницу, в таблице отображаются первые пять записей. Щелчок мышью на кнопке `NextPage` приводит к отображению следующих пяти записей (или, в конце таблицы, оставшегося числа записей), а щелчок на кнопке `PreviousPage` вызывает отображение предыдущих пяти записей (или, в начале таблицы, первых пяти записей). Щелчок на кнопке `FirstPage` или `LastPage` приводит к отображению соответственно первых или последних пяти записей.

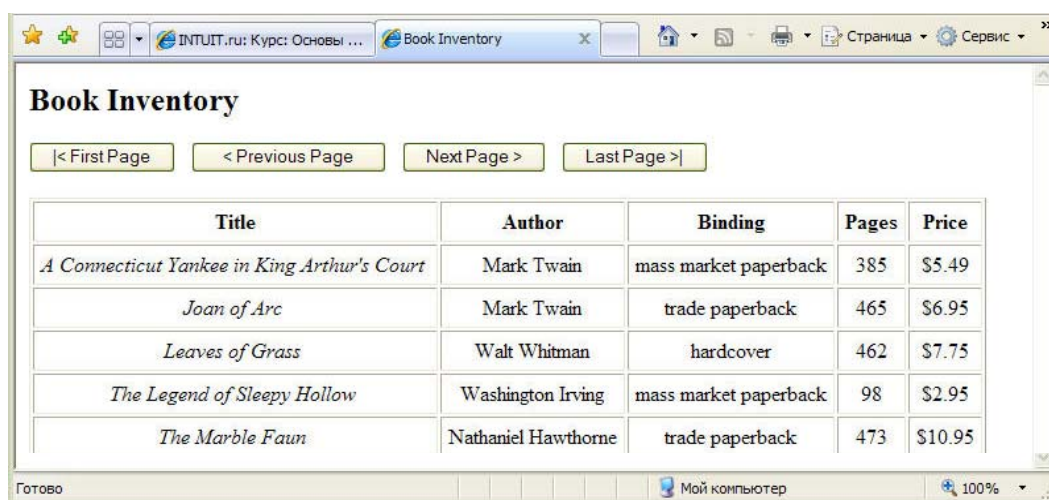


Рис. 8.2. Отображение в Internet Explorer 5 связи данных HTML с XML

Листинг 8.3. `Inventory Big.xml`

```
<?xml version="1.0"?>
<!-- Имя файла: Inventory Big.xml -->
<INVENTORY>
  <BOOK>
```

```

        <TITLE>The Adventures of Huckleberry Finn</TITLE>
        <AUTHOR>Mark Twain</AUTHOR>
        <BINDING>mass market paperback</BINDING>
        <PAGES>298</PAGES>
        <PRICE>$5.49</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>The Adventures of Tom Sawyer</TITLE>
        <AUTHOR>Mark Twain</AUTHOR>
        <BINDING>mass market paperback</BINDING>
        <PAGES>205</PAGES>
        <PRICE>$4.75</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>The Ambassadors</TITLE>
        <AUTHOR>Henry James</AUTHOR>
        <BINDING>mass market paperback</BINDING>
        <PAGES>305</PAGES>
        <PRICE>$5.95</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>The Awakening</TITLE>
        <AUTHOR>Kate Chopin</AUTHOR>
        <BINDING>mass market paperback</BINDING>
        <PAGES>195</PAGES>
        <PRICE>$4.95</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>Billy Budd</TITLE>
        <AUTHOR>Herman Melville</AUTHOR>
        <BINDING>mass market paperback</BINDING>
        <PAGES>195</PAGES>
        <PRICE>$4.49</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>A Connecticut Yankee in King Arthur's
Court</TITLE>
        <AUTHOR>Mark Twain</AUTHOR>
        <BINDING>mass market paperback</BINDING>
        <PAGES>385</PAGES>
        <PRICE>$5.49</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>Joan of Arc</TITLE>
        <AUTHOR>Mark Twain</AUTHOR>
        <BINDING>trade paperback</BINDING>

```



```

        <PAGES>465</PAGES>
        <PRICE>$6.95</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>Leaves of Grass</TITLE>
        <AUTHOR>Walt Whitman</AUTHOR>
        <BINDING>hardcover</BINDING>
        <PAGES>462</PAGES>
        <PRICE>$7.75</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>The Legend of Sleepy Hollow</TITLE>
        <AUTHOR>Washington Irving</AUTHOR>
        <BINDING>mass market paperback</BINDING>
        <PAGES>98</PAGES>
        <PRICE>$2.95</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>The Marble Faun</TITLE>
        <AUTHOR>Nathaniel Hawthorne</AUTHOR>
        <BINDING>trade paperback</BINDING>
        <PAGES>473</PAGES>
        <PRICE>$10.95</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>Moby-Dick</TITLE>
        <AUTHOR>Herman Melville</AUTHOR>
        <BINDING>hardcover</BINDING>
        <PAGES>724</PAGES>
        <PRICE>$9.95</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>Passing</TITLE>
        <AUTHOR>Nella Larsen</AUTHOR>
        <BINDING>trade paperback</BINDING>
        <PAGES>165</PAGES>
        <PRICE>$5.95</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>The Portrait of a Lady</TITLE>
        <AUTHOR>Henry James</AUTHOR>
        <BINDING>mass market paperback</BINDING>
        <PAGES>256</PAGES>
        <PRICE>$4.95</PRICE>
    </BOOK>
    <BOOK>

```



```

<TABLE ID="InventoryTable" DATASRC="#dsoInventory"
DATAPAGESIZE="5" BORDER="1" CELLPADDING="5">
<THEAD>
  <TH>Title</TH>
  <TH>Author</TH>
  <TH>Binding</TH>
  <TH>Pages</TH>
  <TH>Price</TH>
</THEAD>
<TR ALIGN="center">
  <TD><SPAN DATAFLD="TITLE"
  STYLE="font-style:italic"></SPAN></TD>
  <TD><SPAN DATAFLD="AUTHOR"></SPAN></TD>
  <TD><SPAN DATAFLD="BINDING"></SPAN></TD>
  <TD><SPAN DATAFLD="PAGES"></SPAN></TD>
  <TD><SPAN DATAFLD="PRICE"></SPAN></TD>
</TR>
</TABLE>
</BODY>
</HTML>

```

8.4.3. Использование вложенных таблиц для отображения иерархической структуры записей

В предыдущих пунктах вы узнали, как использовать одну таблицу для отображения XML-документа, структурированного как простой набор записей, где каждая запись состоит из фиксированного набора полей, каждое из которых содержит только символьные данные. Теперь вы узнаете, как использовать вложенные таблицы для отображения XML-документа, элементы которого структурированы как иерархический набор записей.

В иерархическом наборе записей каждая запись может содержать, в дополнение к фиксированному набору полей, переменное число вхождений (нуль или более) вложенных записей. В листинге 8.5 представлен пример XML-документа, структурированного как иерархический набор записей. В этом документе корневой элемент (INVENTORY) содержит группу записей CATEGORY. Каждая запись CATEGORY начинается с поля CATNAME, которое содержит только символьные данные, а затем следуют нуль или несколько вложенных записей BOOK. Каждая вложенная запись BOOK имеет пять полей (TITLE, AUTHOR, BINDING, PAGES, PRICE).

Листинг 8.5. Inventory Hierarchy.xml

```
<!-- Имя файла: Inventory Hierarchy.xml -->
<INVENTORY>
  <CATEGORY>
    <CATNAME>Middle Ages</CATNAME>
    <BOOK>
      <TITLE>The Canterbury Tales</TITLE>
      <AUTHOR>Geoffrey Chaucer</AUTHOR>
      <BINDING>hardcover</BINDING>
      <PAGES>692</PAGES>
      <PRICE>$18.95</PRICE>
    </BOOK>
    <BOOK>
      <TITLE>Piers Plowman</TITLE>
      <AUTHOR>William Langland</AUTHOR>
      <BINDING>trade paperback</BINDING>
      <PAGES>385</PAGES>
      <PRICE>$10.95</PRICE>
    </BOOK>
  </CATEGORY>
  <CATEGORY>
    <CATNAME>Renaissance</CATNAME>
    <BOOK>
      <TITLE>The Blazing World</TITLE>
      <AUTHOR>Margaret Cavendish</AUTHOR>
      <BINDING>trade paperback</BINDING>
      <PAGES>225</PAGES>
      <PRICE>$8.79</PRICE>
    </BOOK>
    <BOOK>
      <TITLE>Oroonoko</TITLE>
      <AUTHOR>Aphra Behn</AUTHOR>
      <BINDING>mass market paperback</BINDING>
      <PAGES>295</PAGES>
      <PRICE>$4.95</PRICE>
    </BOOK>
    <BOOK>
      <TITLE>Doctor Faustus</TITLE>
      <AUTHOR>Christopher Marlowe</AUTHOR>
      <BINDING>hardcover</BINDING>
      <PAGES>472</PAGES>
      <PRICE>$15.95</PRICE>
    </BOOK>
  </CATEGORY>
  <CATEGORY>
    <CATNAME>18th Century</CATNAME>
```

```

<BOOK>
  <TITLE>Gulliver's Travels</TITLE>
  <AUTHOR>Jonathan Swift</AUTHOR>
  <BINDING>hardcover</BINDING>
  <PAGES>324</PAGES>
  <PRICE>$11.89</PRICE>
</BOOK>
<BOOK>
  <TITLE>The History of Tom Jones: A Foundling</TITLE>
  <AUTHOR>Henry Fielding</AUTHOR>
  <BINDING>hardcover</BINDING>
  <PAGES>438</PAGES>
  <PRICE>$16.95</PRICE>
</BOOK>
<BOOK>
  <TITLE>Love in Excess</TITLE>
  <AUTHOR>Eliza Haywood</AUTHOR>
  <BINDING>trade paperback</BINDING>
  <PAGES>429</PAGES>
  <PRICE>$12.95</PRICE>
</BOOK>
<BOOK>
  <TITLE>Tristram Shandy</TITLE>
  <AUTHOR>Laurence Sterne</AUTHOR>
  <BINDING>hardcover</BINDING>
  <PAGES>322</PAGES>
  <PRICE>$9.49</PRICE>
</BOOK>
</CATEGORY>
<CATEGORY>
  <CATNAME>19th Century</CATNAME>
  <BOOK>
    <TITLE>Dracula</TITLE>
    <AUTHOR>Bram Stoker</AUTHOR>
    <BINDING>hardcover</BINDING>
    <PAGES>395</PAGES>
    <PRICE>$17.95</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>Great Expectations</TITLE>
    <AUTHOR>Charles Dickens</AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>639</PAGES>
    <PRICE>$6.95</PRICE>
  </BOOK>
  <BOOK>

```

```

        <TITLE>Percival Keene</TITLE>
        <AUTHOR>Frederick Marryat</AUTHOR>
        <BINDING>trade paperback</BINDING>
        <PAGES>425</PAGES>
        <PRICE>$12.89</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>Treasure Island</TITLE>
        <AUTHOR>Robert Louis Stevenson</AUTHOR>
        <BINDING>trade paperback</BINDING>
        <PAGES>283</PAGES>
        <PRICE>$11.85</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>Wuthering Heights</TITLE>
        <AUTHOR>Emily Bronte</AUTHOR>
        <BINDING>hardcover</BINDING>
        <PAGES>424</PAGES>
        <PRICE>$12.95</PRICE>
    </BOOK>
</CATEGORY>
</INVENTORY>

```

Листинг 8.6 содержит HTML-страницу, которая использует вложенную таблицу для отображения иерархической структуры записей XML-документа из листинга 8.5.

Листинг 8.6. Inventory Hierarchy.htm

```

<!-- Имя файла: Inventory Hierarchy.htm -->
<HTML>
<HEAD>
    <TITLE>Inventory of Classic English Literature</TITLE>
</HEAD>
<BODY>
<XML ID="dsoInventory" SRC="Inventory Hierarchy.xml"></XML>
    <TABLE DATASRC="#dsoInventory" BORDER="1">
        <THEAD>
            <TH>Classic English Literature</TH>
        </THEAD>
        <TR>
            <TD><SPAN DATAFLD="CATNAME"></SPAN></TD>
        </TR>
        <TR>
            <TD>
                <TABLE DATASRC="#dsoInventory" DATAFLD="BOOK"
                    BORDER="0" CELLSPACING="10">

```

```

        <THEAD>
            <TH>Title</TH>
            <TH>Author</TH>
            <TH>Binding</TH>
            <TH>Pages</TH>
            <TH>Price</TH>
        </THEAD>
        <TR ALIGN="CENTER">
            <TD><SPAN DATAFLD="TITLE"
            STYLE="font-style:italic"></SPAN></TD>
            <TD><SPAN DATAFLD="AUTHOR"></SPAN></TD>
            <TD><SPAN DATAFLD="BINDING"></SPAN></TD>
            <TD><SPAN DATAFLD="PAGES"></SPAN></TD>
            <TD><SPAN DATAFLD="PRICE"></SPAN></TD>
        </TR>
    </TABLE>
</TD>
</TR>
</TABLE>
</BODY>
</HTML>

```

В листинге 8.6 внешняя таблица сцеплена с XML-документом, что видно из описания начального тега:

```
<TABLE DATASRC="#dsoInventory" BORDER="1">
```

Внешняя таблица также включает заголовок (элемент THEAD, отображающий надпись «Classic English Literature»), плюс две строки таблицы (два элемента TR). Браузер повторяет две строки для каждой записи верхнего уровня (т. е. для каждой записи CATEGORY). В первой строке отображается поле CATNAME. Это подобно рассмотренному ранее в листинге 8.2 (см. на с. 211–212) примеру таблицы, отображающей простой набор записей. Однако вторая строка не отображает поле, а содержит вложенную таблицу, которая отображает содержимое каждой вложенной записи BOOK внутри текущей категории. Вот разметка для вложенной таблицы:

```

    <TABLE DATASRC="#dsoInventory" DATAFLD="BOOK" BORDER=0
    CELLSPACING=10>
        <THEAD>
            <TH>Title</TH>
            <TH>Author</TH>
            <TH>Binding</TH>

```

```

<TH>Pages</TH>
<TH>Price</TH>
</THEAD>
<TR ALIGN="CENTER">
  <TD><SPAN DATAFLD="TITLE"
    STYLE="font-style:italic"></SPAN></TD>
  <TD><SPAN DATAFLD="AUTHOR"></SPAN></TD>
  <TD><SPAN DATAFLD="BINDING"></SPAN></TD>
  <TD><SPAN DATAFLD="PAGES"></SPAN></TD>
  <TD><SPAN DATAFLD="PRICE"></SPAN></TD>
</TR>
</TABLE>

```

Обратите внимание, что вы должны сцепить вложенную таблицу не только с XML-документом (DATASRC= «#dsoInventory»), но и с вложенными записями BOOK (DATAFLD= «BOOK»), чтобы в таблице отображалось содержимое каждой записи BOOK, вложенной в текущую запись CATEGORY. Другими словами, строковый элемент (TR) в этой таблице будет повторен для каждого из этих элементов BOOK (рис. 8.3). (Заметим, что внешняя таблица по умолчанию сцеплена с записями верхнего уровня (в данном случае с записями CATEGORY), поэтому каждая из этих записей отображается при переходе к новой категории.)

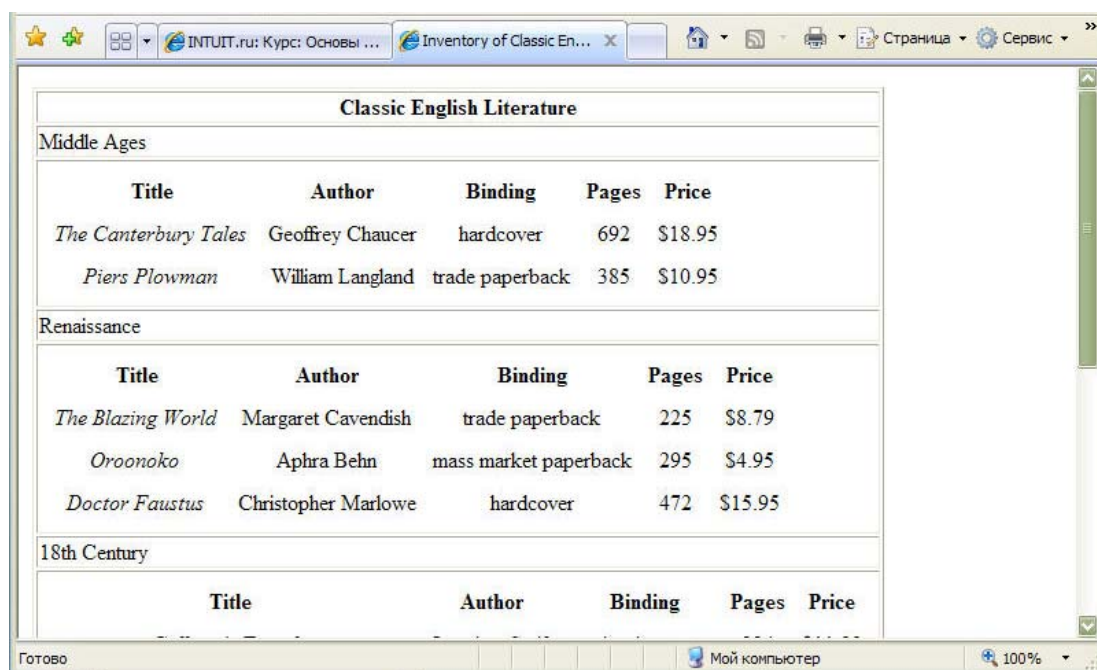


Рис. 8.3. Отображение в Internet Explorer 5 содержимого записи BOOK

Вы можете использовать дополнительные вложенные таблицы для отображения документа, который содержит дополнительные уровни вложенных записей. Рассмотрим в качестве примера документ из листинга 8.5. Предположим, что вы заменили каждое поле AUTHOR:

```
<AUTHOR>Geoffrey Chaucer</AUTHOR>
```

на нуль или несколько записей AUTHOR:

```
<AUTHOR>  
  <FIRSTNAME>Geoffrey</FIRSTNAME>  
  <LASTNAME>Chaucer</LASTNAME>  
</AUTHOR>
```

В этом случае вы можете использовать дополнительную вложенную таблицу для отображения всех авторов для каждого элемента BOOK, применив ту же технику, которую вы использовали для одиночной вложенной таблицы.

8.5. ИСПОЛЬЗОВАНИЕ СВЯЗЫВАНИЯ ДАННЫХ ПО ОДНОЙ ЗАПИСИ

Связывание данных по одной записи применяется для HTML-элементов, которые не являются таблицами и не включены в связанную таблицу. HTML-элемент (например, SPAN, BUTTON или LABEL) связывается с отдельным полем XML. После этого HTML-элемент автоматически отображает содержимое поля XML, с которым он связан (сцеплен). Например, следующий HTML-элемент SPAN сцеплен с полем TITLE XML-документа, доступ к которому осуществляется через фрагмент данных dsoBook:

```
<SPAN DATASRC="#dsoBook" DATAFLD="TITLE"></SPAN>
```

Поскольку HTML-элемент не имеет множественных частей, подобно таблице, он способен отобразить значение поля только для одной записи за раз. Чтобы использовать связывание данных по одной записи, XML-документ должен быть организован как простой набор записей.

Наипростейшим способом связывания данных по одной записи является случай, когда XML-документ состоит только из одной записи, подобно документу из листинга 8.7.

Листинг 8.7. Book.xml

```
<?xml version="1.0"?>
<!-- Имя файла: Book.xml -->
<BOOK>
  <TITLE>The Adventures of Huckleberry Finn</TITLE>
  <AUTHOR>Mark Twain</AUTHOR>
  <BINDING>mass market paperback</BINDING>
  <PAGES>298</PAGES>
  <PRICE>$5.49</PRICE>
</BOOK>
```

В листинге 8.8 представлена HTML-страница, которая связывает отдельный элемент SPAN с каждым из полей рассматриваемого документа (TITLE, AUTHOR, BINDING, PAGES, PRICE) (рис. 8.4).

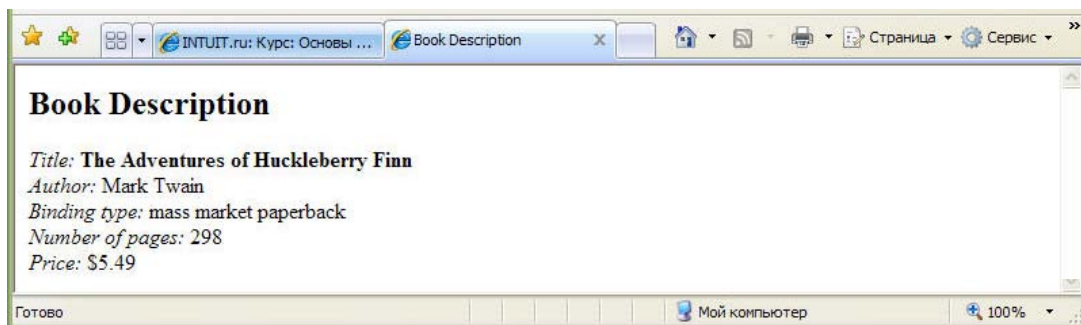


Рис. 8.4. Отображение в Internet Explorer 5 связи элемента SPAN с полями HTML-страницы

Листинг 8.8. Book.htm

```
<!-- Имя файла: Book.htm -->
<HTML>
<HEAD>
  <TITLE>Book Description</TITLE>
</HEAD>
<BODY>
  <XML ID="dsoBook" SRC="Book.xml"></XML>
  <H2>Book Description</H2>
  <SPAN STYLE="font-style:italic">Title: </SPAN>
  <SPAN STYLE="font-weight:bold" DATASRC="#dsoBook"
DATAFLD="TITLE"></SPAN>
  <BR>
  <SPAN STYLE="font-style:italic">Author: </SPAN>
  <SPAN DATASRC="#dsoBook" DATAFLD="AUTHOR"></SPAN>
  <BR>
  <SPAN STYLE="font-style:italic">Binding type: </SPAN>
  <SPAN DATASRC="#dsoBook" DATAFLD="BINDING"></SPAN>
```

```

<BR>
<SPAN STYLE="font-style:italic">Number of pages: </SPAN>
<SPAN DATASRC="#dsoBook" DATAFLD="PAGES"></SPAN>
<BR>
<SPAN STYLE="font-style:italic">Price: </SPAN>
<SPAN DATASRC="#dsoBook" DATAFLD="PRICE"></SPAN>
</BODY>
</HTML>

```

8.5.1. Перемещение между записями

Если XML-документ содержит более одной записи (как большинство из них), связывание данных по записям становится несколько более сложным, поскольку HTML-элемент может отобразить за раз только одну запись. Отображаемая в данный момент запись называется текущей. (Связывание данных по одной записи иногда называют еще связыванием по текущей записи.) Изначально текущей является первая запись в документе.

DSO (объект исходных данных), ассоциированный с XML-документом, предоставляет ряд методов (функций), которыми вы можете воспользоваться при перемещении между записями. Эти методы принадлежат объекту `recordset` DSO и приведены в табл. 8.2. Заметим, что примеры вызовов, представленные в последнем столбце, предполагают, что HTML-страница содержит фрагмент данных XML с идентификатором (ID) `dsoInventory`.

Таблица 8.2

Методы объекта `recordset` DSO для перемещения между записями

Метод объекта <code>recordset</code> DSO	Эффект	Пример вызова
<code>moveFirst</code>	Переход от текущей записи к первой записи в документе	<code>dsoInventory.recordset.moveFirst()</code>
<code>movePrevious</code>	Переход от текущей записи к предыдущей записи	<code>dsoInventory.recordset.movePrevious()</code>
<code>moveNext</code>	Переход от текущей записи к следующей записи	<code>dsoInventory.recordset.moveNext()</code>
<code>moveLast</code>	Переход от текущей записи к последней записи в документе	<code>dsoInventory.recordset.moveLast()</code>
<code>move</code>	Переход от текущей записи к записи с указанным номером	<code>dsoInventory.recordset.move(5)</code> (Переход к пятой записи. Записи нумеруются, начиная с нуля.)



Примечание. Составной объект recordset DSO соответствует стандарту технологии доступа к данным, которую Microsoft назвала ActiveX Data Objects (ADO). Вы можете использовать объект общего назначения ADO recordset совместно с множеством различных источников данных, а не только с XML DSO. Вы можете обращаться к этим методам из написанного вами кода сценария. Однако самый простой способ их вызова – это присвоить имя метода атрибуту ONCLICK элемента BUTTON, как в следующем примере:

```
<BUTTON ONCLICK="dsoInventory.recordset.moveFirst()">
  First Record
</BUTTON>
```

Этот элемент отображает кнопку. Когда пользователь щелкает мышью на кнопке, вызывается метод, присвоенный атрибуту ONCLICK, dsoInventory.recordset.moveFirst.

Если текущей является первая запись, вызов метода movePrevious приводит к перемещению в зону начала файла (BOF), где нет записей, поэтому сцепленный элемент будет пуст. Аналогично, вызов метода moveNext, если текущей является последняя запись, вызывает перемещение в зону конца файла (EOF), поэтому сцепленный элемент также будет пуст.

К счастью, объект recordset поддерживает свойство BOF, которое принимает значение true (истина), если достигнуто начало файла, а также свойство EOF, которое принимает значение true (истина), если достигнут конец файла. Вы можете использовать эти свойства для определения этих состояний и внесения необходимых корректировок. Например, приведенный ниже код предписывает при щелчке на кнопке в случае, если достигнуто начало файла, быстро отобразить первую запись:

```
<BUTTON ONCLICK="dsoInventory.recordset.movePrevious();
  if (dsoInventory.recordset.BOF)
    dsoInventory.recordset.moveNext()">
Back
</BUTTON>
```

Следующий код проверяет достижение конца файла:

```
<BUTTON ONCLICK="dsoInventory.recordset.moveNext();
  if (dsoInventory.recordset.EOF)
    dsoInventory.recordset.movePrevious()">
Forward
</BUTTON>
```

Обратите внимание, что вы можете присвоить атрибуту ONCLICK целый блок кода сценария. В этих примерах код написан на языке Microsoft JScript. Далее в этой главе вы узнаете, как писать самостоятельные фрагменты кодов сценариев, которые позволяют включать значительно большее число инструкций.

В следующем упражнении вы создадите HTML-страницу, которая отображает XML-документ из листинга 8.3 (см. на с. 215–218) по одной записи за раз. Страница содержит кнопки для перехода к первой, предыдущей, последующей и последней записям.

Листинг 8.9. Inventory Single.htm

```
<!-- Имя файла: Inventory Single.htm -->
<HTML>
<HEAD>
  <TITLE>Book Inventory</TITLE>
</HEAD>
<BODY>
  <XML ID="dsoInventory" SRC="Inventory Big.xml"></XML>
  <H2>Book Description</H2>
  <SPAN STYLE="font-style:italic">Title: </SPAN>
  <SPAN DATASRC="#dsoInventory" DATAFLD="TITLE"
STYLE="font-weight:bold"></SPAN>
  <BR>
  <SPAN STYLE="font-style:italic">Author: </SPAN>
  <SPAN DATASRC="#dsoInventory" DATAFLD="AUTHOR"></SPAN>
  <BR>
  <SPAN STYLE="font-style:italic">Binding type: </SPAN>
  <SPAN DATASRC="#dsoInventory" DATAFLD="BINDING"></SPAN>
  <BR>
  <SPAN STYLE="font-style:italic">Number of pages: </SPAN>
  <SPAN DATASRC="#dsoInventory" DATAFLD="PAGES"></SPAN>
  <BR>
  <SPAN STYLE="font-style:italic">Price: </SPAN>
  <SPAN DATASRC="#dsoInventory" DATAFLD="PRICE"></SPAN>
  <HR>
  <BUTTON ONCLICK="dsoInventory.recordset.moveFirst()">
    |&lt; First
  </BUTTON>
  <BUTTON ONCLICK="dsoInventory.recordset.movePrevious();
    if (dsoInventory.recordset.BOF)
      dsoInventory.recordset.moveNext()">
    &lt; Back
  </BUTTON>
  <BUTTON ONCLICK="dsoInventory.recordset.moveNext();
    if (dsoInventory.recordset.EOF)
```

```

        dsoInventory.recordset.movePrevious() ">
        Forward &gt;
    </BUTTON>
    <BUTTON ONCLICK="dsoInventory.recordset.moveLast() ">
        Last &gt;|
    </BUTTON>
</BODY>
</HTML>

```

В Windows Explorer (Проводник) или в окне папки дважды щелкните на имени файла *Inventory Single.htm*, который вы сохранили на предыдущем шаге, и получите то, что изображено на рис. 8.5.

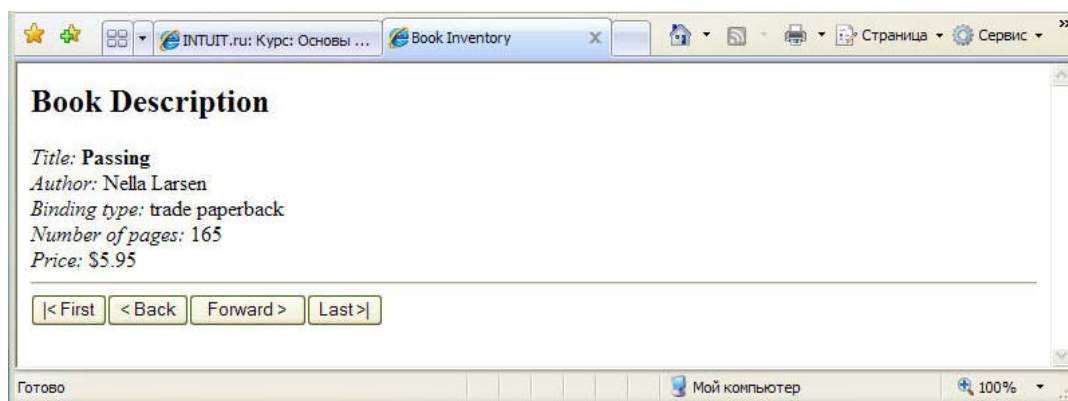


Рис. 8.5. Отображение в Internet Explorer 5 файла *Inventory Single.htm*

Заметим, что изначально, пока пользователь еще не щелкнул мышью на какой-либо кнопке, Internet Explorer 5 отображает только первую запись в документе.

8.5.2. Другие способы связывания данных

В последующих пунктах вы познакомитесь с рядом других способов для связывания нетабличных HTML-элементов. Это могут быть как индивидуальные HTML-элементы, используемые для связывания данных по одной записи, так и HTML-элементы, содержащиеся в сцепленной таблице HTML. Конкретно вы узнаете:

- 1) как связывать другие HTML-элементы с полями XML;
- 2) как воспроизводить HTML-разметку, содержащуюся в полях XML;
- 3) как обновлять имеющиеся данные XML.

В табл. 8.3 представлена важная информация, которая вам потребуется при изучении материала последующих пунктов. В ней приведены HTML-элементы, которые вы можете использовать для связывания данных по одной записи, т. е. все сцепляемые HTML-элементы, за исключением элемента TABLE. Для каждого элемента описано его назначение, оговорено свойство элемента, через которое он сцепляется с полем XML, указано, может ли элемент передавать HTML-разметку, содержащуюся в поле XML, с которым он сцеплен, а также может ли элемент обновлять содержимое поля XML.

Таблица 8.3

Сцепляемые HTML-элементы (за исключением элемента TABLE)

HTML-элемент	Назначение элемента	Свойство	Передаёт разметку HTML	Обновляет сцепленное поле XML
1	2	3	4	5
A	Элемент «анкер» указывает на начало или место назначения гиперссылки	href	Нет	Нет
APPLET	Добавляет на страницу фрагмент кода Java	param	Нет	Да
BUTTON	Отображает кнопку управления	innerHTML и innerText	Да	Нет
DIV	Используется для форматирования части документа, такой как глава, раздел или приложение	innerHTML и innerText	Да	Нет
FRAME	Содержит отдельный фрейм (кадр) в наборе фреймов	src	Нет	Нет
IFRAME	Создает невыделенный объектный фрейм	src	Нет	Нет
IMG	Отображает рисунок или видеоклип	src	Нет	Нет
INPUT TYPE=CHECKBOX	Отображает поле флажка в форме	checked	Нет	Да
INPUT TYPE=HIDDEN	Хранит и предоставляет информацию на сервер вместо отображения в форме	value	Нет	Да
INPUT TYPE=PASSWORD	Аналогично INPUT TYPE=TEXT, но вводимый пользователем текст не отображается	value	Нет	Да
INPUT TYPE=RADIO	Отображает радиокнопку в форме	checked	Нет	Да

1	2	3	4	5
INPUT TYPE=TEXT	Разрешает пользователю ввести однострочный текст в форме	value	Нет	Да
LABEL	Отображает текстовую надпись (этикетку)	innerText и innerHTML	Да	Нет
MARQUEE	Отображает текст с прокруткой	innerText и innerHTML	Да	Нет
SELECT	Отображает поле списка	Свойство text для выбранного элемента списка	Нет	Да
SPAN	Используется для форматирования фрагмента текста (например, текста внутри элемента P [абзац] или DIV)	innerText и innerHTML	Да	Нет
TEXTAREA	Разрешает пользователю ввести многострочный текст	value	Нет	Да

8.5.3. Связывание с другими HTML-элементами

При связывании элемента SPAN с полем XML элемент просто отображает содержимое поля. Это происходит потому, что свойство innerText элемента SPAN, которое определяет текст, отображаемый элементом, сцеплено с полем XML.

⇒ *Примечание.* В DHTML, поддерживаемом Internet Explorer 5, каждый HTML-элемент обладает набором свойств, которые вы можете использовать для установки или извлечения различных характеристик элемента с помощью кода сценария. Кроме того, как пояснено в этом пункте, для свойства автоматически устанавливается значение поля XML, с которым оно сцеплено. Однако для некоторых сцепляемых HTML-элементов с полем XML сцепляются и другие свойства.

⇒ *Примечание.* При связывании с элементом SPAN реально сцепляются его свойства innerText и innerHTML. Свойство innerText устанавливает или получает текстовое содержимое элемента, не включая в него HTML-разметку. Свойство innerHTML устанавливает или получает полное содержимое элемента, включая любую HTML-разметку.

Например, для следующего сцепленного элемента А (элемент «анкер», используемый для создания гиперссылок) свойство href сцеплено с полем XML:

```
<A DATASRC="dsoInventory" DATAFLD="REVIEWS">
  Click here for reviews
</A>
```

Это свойство, как и атрибут HREF элемента, устанавливает URL для гиперссылки. Следовательно, из поля XML извлекается URL гиперссылки для сцепленного элемента А, а не его текстовое содержимое.

В качестве другого примера рассмотрим связывание поля флажка элемента INPUT через свойство checked (которое изменяет статус выбора элемента (установлен или сброшен флажок)) с XML-полем:

```
<INPUT TYPE="CHECKBOX" DATASRC="#dsoInventory"
DATAFLD="INSTOCK">
```

Если XML-поле INSTOCK пусто, либо содержит текст «0» или «false», то поле флажка очищается. Если оно содержит какой-либо другой текст, флажок устанавливается.

В качестве последнего примера рассмотрим элемент IMG (изображение), свойство src которого сцеплено с XML-полем. Это свойство, как и атрибут SRC элемента, задает URL файла, содержащего графические данные. Листинги 8.10 и 8.11 иллюстрируют технику сцепления для элемента IMG.

Листинг 8.10. Inventory Image.xml

```
<?xml version="1.0"?>
<!-- Имя файла: Inventory Image.xml -->
<INVENTORY>
  <BOOK>
    <COVERIMAGE>Leaves.bmp</COVERIMAGE>
    <TITLE>Leaves of Grass</TITLE>
    <AUTHOR>Walt Whitman</AUTHOR>
    <BINDING>hardcover</BINDING>
    <PAGES>462</PAGES>
    <PRICE>$7.75</PRICE>
  </BOOK>
  <BOOK>
    <COVERIMAGE>Legend.bmp</COVERIMAGE>
    <TITLE>The Legend of Sleepy Hollow</TITLE>
    <AUTHOR>Washington Irving</AUTHOR>
    <BINDING>mass market paperback</BINDING>
```

```

        <PAGES>98</PAGES>
        <PRICE>$2.95</PRICE>
    </BOOK>
    <BOOK>
        <COVERIMAGE>Moby.bmp</COVERIMAGE>
        <TITLE>Moby-Dick</TITLE>
        <AUTHOR>Herman Melville</AUTHOR>
        <BINDING>hardcover</BINDING>
        <PAGES>724</PAGES>
        <PRICE>$9.95</PRICE>
    </BOOK>
</INVENTORY>

```

Листинг 8.11. Inventory Image Table.htm

```

<!-- Имя файла: Inventory Image Table.htm -->
<HTML>
<HEAD>
    <TITLE>Book Inventory</TITLE>
</HEAD>
<BODY>
    <XML ID="dsoInventory" SRC="Inventory Image.xml"></XML>
    <H2>Book Inventory</H2>
    <TABLE DATASRC="#dsoInventory" BORDER="1" CELLPADDING="5">
        <THEAD>
            <TH>Cover</TH>
            <TH>Title</TH>
            <TH>Author</TH>
            <TH>Binding</TH>
            <TH>Pages</TH>
            <TH>Price</TH>
        </THEAD>
        <TR ALIGN="center">
            <TD><IMG DATAFLD="COVERIMAGE"></TD>
            <TD><SPAN DATAFLD="TITLE"
                STYLE="font-style:italic"></SPAN></TD>
            <TD><SPAN DATAFLD="AUTHOR"></SPAN></TD>
            <TD><SPAN DATAFLD="BINDING"></SPAN></TD>
            <TD><SPAN DATAFLD="PAGES"></SPAN></TD>
            <TD><SPAN DATAFLD="PRICE"></SPAN></TD>
        </TR>
    </TABLE>
</BODY>
</HTML>

```

Листинг 8.10 есть XML-документ, который содержит поле с именем COVERIMAGE в каждой записи BOOK. Каждое поле COVERIMAGE содержит URL графического файла, который хранит рисунок обложки книги. Листинг 8.11 – это та же самая HTML-страница, что и в листинге 8.2, за исключением того, что в начало каждой строки таблицы добавлена дополнительная ячейка (элемент TD), которая содержит элемент IMG, а не SPAN. Элемент IMG сцеплен с полем COVERIMAGE XML-документа, и поэтому отобразит рисунок обложки для каждой книги.

Вы можете поэкспериментировать со связыванием некоторых других HTML-элементов, содержащихся в табл. 8.3 (см. на с. 231–232), чтобы познакомиться с их сцепляемыми свойствами и способами использования элементами данных, предоставляемых XML-полями, с которыми они сцеплены.

8.6. ПЕРЕДАЧА HTML-РАЗМЕТКИ

По умолчанию, если символьные данные XML-поля включают HTML-разметку, то HTML-элемент, сцепленный с этим полем, воспринимает и отображает символы разметки как литералы. Рассмотрим, например, следующий элемент SPAN, который сцеплен с XML-полем AUTHOR-BIO:

```
<SPAN DATASRC="#dsoInventory" DATAFLD="AUTHOR-BIO"></SPAN>
```

Если поле AUTHOR-BIO содержит элемент I (курсив), например:

```
<AUTHOR-BIO>Henry James was an American author who  
lived from 1843 to 1916, and wrote<I>The Bostonians</I>  
and many other works of psychologically realistic  
fiction. </AUTHOR-BIO>
```

элемент SPAN, который воспринимает символы HTML-разметки как литералы, отобразит содержимое поля следующим образом:

```
Henry James was an American author who lived from  
1843 to 1916, and wrote <I>The Bostonians</I> and many  
other works of psychologically realistic fiction.
```

Для некоторых сцепляемых HTML-элементов, таких как SPAN, вы можете установить для атрибута DATAFORMATAS значение «HTML», что заставит браузер обрабатывать любую HTML-разметку, содержащуюся в тексте поля, а не просто воспринимать ее

как символы-литералы. Предположим, вы определили рассмотренный ранее элемент SPAN следующим образом:

```
<SPAN DATASRC="#dsoInventory" DATAFLD="AUTHOR-BIO"  
DATAFORMATAS="HTML"></SPAN>
```

Текст внутри элемента I будет воспринят как текст с курсивным начертанием:

Henry James was an American author who lived from 1843 to 1916, and wrote The Bostonians and many other works of psychologically realistic fiction.

⇒ *Примечание.* Присвоение атрибуту DATAFORMATAS его значения по умолчанию («TEXT») дает тот же эффект, что и пропуск этого атрибута – символы HTML-разметки будут восприниматься как литералы.

Чтобы узнать, какие элементы вы можете использовать для передачи HTML-разметки посредством установки атрибута DATAFORMATAS= «HTML», обратитесь к табл. 8.3 (см. на с. 231–232). Для таких элементов в предпоследнем столбце таблицы («Передает разметку HTML») стоит «Да».

Вставка и передача HTML-разметки в XML-поля весьма полезна для изменения формата части текста (например, с использованием элементов I или B) и для включения HTML-элементов, таких как гиперссылки или изображения, в текст. Хотя форматирование XML-текста путем включения в XML HTML-разметки нарушает принцип разделения данных и форматов, при связывании данных эта техника является единственным подходящим способом модификации формата или включения HTML-элементов внутрь поля. (При использовании других методов отображения XML, рассматриваемых в этом курсе, вы обычно имеете возможность форматировать или вставлять элементы внутрь XML-элемента путем включения дочерних элементов и соответствующей их обработки.)

Когда вы добавляете HTML-разметку в XML-поле, вы не можете помещать в текст символ левой угловой скобки (<) или знак амперсанда (&) как литералы. (Напомним, что эти символы недопустимы в символьных данных элемента.) Однако вы можете вставлять их с использованием предопределенных ссылок на примитивы < и &. Другой возможностью сделать HTML-разметку более наглядной, что особенно полезно для большого фрагмента HTML, является использование разделов CDATA.

8.7. ОБНОВЛЕНИЕ НАКОПЛЕННЫХ ДАННЫХ XML

Объект DSO XML дает вам возможность модифицировать данные XML несколькими способами. Прежде, чем начать применять эту возможность, следует представлять, что при этом модифицируется только копия данных XML, которую DSO временно хранит в памяти, а не оригинальный XML-документ на сервере. Если вы не используете известные способы обновления оригинального документа на сервере (они не рассматриваются в этом курсе), обновление одних лишь накопленных данных XML несет мало пользы.

Вы можете разрешить пользователю модифицировать определенное XML-поле, сцепив его с HTML-элементом, допускающим обновление, например элемент INPUT типа TEXT. В крайнем правом столбце табл. 8.3 содержатся сведения, допускает ли HTML-элемент обновление пользователем XML-поля, сцепленного с этим элементом.

Например, если вместо сцепления поля TITLE с элементом SPAN вы свяжете его с элементом INPUT типа TEXT, как показано ниже, пользователь сможет редактировать, а не только просматривать содержимое TITLE:

```
<INPUT TYPE="TEXT" DATASRC="#dsoInventory" DATAFLD="TITLE">
```

Кроме того, объект recordset DSO предоставляет методы, которые позволяют вам добавлять или удалять целые записи из накопленного набора записей, а также отменять модификацию записей. Эти методы сведены в табл. 8.4. Заметим, что для примеров в последнем столбце предполагается, что HTML-страница содержит фрагмент данных XML с идентификатором (ID) dsoInventory.

Таблица 8.4

Методы объекта recordset DSO для добавления и удаления записей

Метод объекта recordset DSO	Эффект	Пример обращения
addNew	Добавляет новую запись к накопленному набору записей	dsoInventory.recordset.addNew()
delete	Удаляет текущую запись из накопленного набора записей	dsoInventory.recordset.delete()
cancelUpdate	Возвращает любые изменения, сделанные для полей текущей записи, либо удаляет вновь введенную запись	dsoInventory.recordset.cancelUpdate()

8.8. ИСПОЛЬЗОВАНИЕ DTD ПРИ СВЯЗЫВАНИИ ДАННЫХ

Все рассматриваемые ранее в этой главе документы были корректно сформированными XML-документами без объявления типа документа (DTD). Однако в случае, если вы собираетесь отображать XML-документ с использованием связывания данных, включение DTD и превращение документа в валидный поможет вам обеспечить требуемую симметричность организации набора записей при определении элементов документа. DTD также обеспечивает соответствие свойств DSO для накопленного набора записей и элементов в вашем документе.

Включение DTD особенно полезно для документа, который имеет более сложную иерархическую структуру записей, чем та, которую вы можете отобразить с помощью вложенных таблиц. Примером является документ из листинга 8.5 (см. на с. 220–222). В следующем упражнении вы добавите DTD в этот документ, превратив его в валидный и обеспечив тем самым необходимую для связывания данных структуру записей.

Заметим, что если XML-документ, который вы отображаете с помощью связывания данных, содержит ошибки валидности, в сцепленных элементах не будут отображены никакие данные, но при этом не появится и сообщение об ошибке. Чтобы увидеть сообщение об ошибке в связанном XML-документе, следует протестировать этот документ с применением сценария проверки на корректность и валидность (см. пункт «Проверка валидности XML-документа» (см. на с. 291) в главе 9).



Совет. Когда вы создаете объявление элемента для записи (такой как CATEGORY или BOOK в листинге 8.12), вы должны включить модель содержимого, которая исчерпывающе описывает все поля записи и вложенные записи. При этом нельзя использовать спецификацию содержимого ANY, что приведет к нарушению связей между данными.

Листинг 8.12. Inventory Hierarchy Valid.xml

```
<?xml version="1.0"?>
<!-- Имя файла: Inventory Hierarchy Valid.xml -->
<!DOCTYPE INVENTORY
[
  <!ELEMENT INVENTORY (CATEGORY*)>
  <!ELEMENT CATEGORY (CATNAME, BOOK*)>
]
```

```

<!ELEMENT CATNAME (#PCDATA)>
<!ELEMENT BOOK (TITLE, AUTHOR, BINDING, PAGES, PRICE)>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT AUTHOR (#PCDATA)>
<!ELEMENT BINDING (#PCDATA)>
<!ELEMENT PAGES (#PCDATA)>
<!ELEMENT PRICE (#PCDATA)>
]
>
<INVENTORY>
  <CATEGORY>
    <CATNAME>Middle Ages</CATNAME>
    <BOOK>
      <TITLE>The Canterbury Tales</TITLE>
      <AUTHOR>Geoffrey Chaucer</AUTHOR>
      <BINDING>hardcover</BINDING>
      <PAGES>692</PAGES>
      <PRICE>$18.95</PRICE>
    </BOOK>
    <BOOK>
      <TITLE>Piers Plowman</TITLE>
      <AUTHOR>William Langland</AUTHOR>
      <BINDING>trade paperback</BINDING>
      <PAGES>385</PAGES>
      <PRICE>$10.95</PRICE>
    </BOOK>
  </CATEGORY>
  <CATEGORY>
    <CATNAME>Renaissance</CATNAME>
    <BOOK>
      <TITLE>The Blazing World</TITLE>
      <AUTHOR>Margaret Cavendish</AUTHOR>
      <BINDING>trade paperback</BINDING>
      <PAGES>225</PAGES>
      <PRICE>$8.79</PRICE>
    </BOOK>
    <BOOK>
      <TITLE>Oroonoko</TITLE>
      <AUTHOR>Aphra Behn</AUTHOR>
      <BINDING>mass market paperback</BINDING>
      <PAGES>295</PAGES>
      <PRICE>$4.95</PRICE>
    </BOOK>
    <BOOK>
      <TITLE>Doctor Faustus</TITLE>
      <AUTHOR>Christopher Marlowe</AUTHOR>

```

```

        <BINDING>hardcover</BINDING>
        <PAGES>472</PAGES>
        <PRICE>$15.95</PRICE>
    </BOOK>
</CATEGORY>
<CATEGORY>
    <CATNAME>18th Century</CATNAME>
    <BOOK>
        <TITLE>Gulliver's Travels</TITLE>
        <AUTHOR>Jonathan Swift</AUTHOR>
        <BINDING>hardcover</BINDING>
        <PAGES>324</PAGES>
        <PRICE>$11.89</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>The History of Tom Jones: A Foundling</TITLE>
        <AUTHOR>Henry Fielding</AUTHOR>
        <BINDING>hardcover</BINDING>
        <PAGES>438</PAGES>
        <PRICE>$16.95</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>Love in Excess</TITLE>
        <AUTHOR>Eliza Haywood</AUTHOR>
        <BINDING>trade paperback</BINDING>
        <PAGES>429</PAGES>
        <PRICE>$12.95</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>Tristram Shandy</TITLE>
        <AUTHOR>Laurence Sterne</AUTHOR>
        <BINDING>hardcover</BINDING>
        <PAGES>322</PAGES>
        <PRICE>$9.49</PRICE>
    </BOOK>
</CATEGORY>
<CATEGORY>
    <CATNAME>19th Century</CATNAME>
    <BOOK>
        <TITLE>Dracula</TITLE>
        <AUTHOR>Bram Stoker</AUTHOR>
        <BINDING>hardcover</BINDING>
        <PAGES>395</PAGES>
        <PRICE>$17.95</PRICE>
    </BOOK>
    <BOOK>

```



```

        <TITLE>Great Expectations</TITLE>
        <AUTHOR>Charles Dickens</AUTHOR>
        <BINDING>mass market paperback</BINDING>
        <PAGES>639</PAGES>
        <PRICE>$6.95</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>Percival Keene</TITLE>
        <AUTHOR>Frederick Marryat</AUTHOR>
        <BINDING>trade paperback</BINDING>
        <PAGES>425</PAGES>
        <PRICE>$12.89</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>Treasure Island</TITLE>
        <AUTHOR>Robert Louis Stevenson</AUTHOR>
        <BINDING>trade paperback</BINDING>
        <PAGES>283</PAGES>
        <PRICE>$11.85</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>Wuthering Heights</TITLE>
        <AUTHOR>Emily Bronte</AUTHOR>
        <BINDING>hardcover</BINDING>
        <PAGES>424</PAGES>
        <PRICE>$12.95</PRICE>
    </BOOK>
</CATEGORY>
</INVENTORY>

```

В вашем текстовом редакторе откройте страницу Inventory Hierarchy.htm, созданную вами ранее в этой главе. (Этот документ представлен в листинге 8.6 (см. на с. 222–223).)

Измените атрибут SRC фрагмента данных на странице, чтобы он был связан с новым XML-документом, который вы только что создали, а именно, замените

```
<XML ID="dsoInventory" SRC="Inventory Hierarchy.xml"></XML>
```

на

```
<XML ID="dsoInventory" SRC="Inventory Hierarchy
Valid.xml"></XML>
```

Чтобы отразить новое имя файла, которое вы присвоите, измените комментарий в начале страницы

```
<!-- Имя файла: Inventory Hierarchy.htm -->
```

на

```
<!-- Имя файла: Inventory Hierarchy Valid.htm -->
```

Воспользуйтесь командой Save As (Сохранить как) вашего текстового редактора, чтобы сохранить копию модифицированной страницы, задав в качестве имени файла Inventory Hierarchy Valid.htm.

Полная HTML-страница представлена в листинге 8.13.

Листинг 8.13. Inventory Hierarchy Valid.htm

```
<!-- Имя файла: Inventory Hierarchy Valid.htm -->
<HTML>
<HEAD>
  <TITLE>Inventory of Classic English Literature</TITLE>
</HEAD>
<BODY>
  <XML ID="dsoInventory" SRC="Inventory Hierarchy
Valid.xml">
    </XML>
    <TABLE DATASRC="#dsoInventory" BORDER="1">
      <THEAD>
        <TH>Classic English Literature</TH>
      </THEAD>
      <TR>
        <TD><SPAN DATAFLD="CATNAME"></SPAN></TD>
      </TR>
      <TR>
        <TD>
          <TABLE DATASRC="#dsoInventory" DATAFLD="BOOK"
            BORDER=0 CELLSPACING=10>
            <THEAD>
              <TH>Title</TH>
              <TH>Author</TH>
              <TH>Binding</TH>
              <TH>Pages</TH>
              <TH>Price</TH>
            </THEAD>
            <TR ALIGN="CENTER">
              <TD><SPAN DATAFLD="TITLE"
                STYLE="font-style:italic"></SPAN></TD>
              <TD><SPAN DATAFLD="AUTHOR"></SPAN></TD>
              <TD><SPAN DATAFLD="BINDING"></SPAN></TD>
              <TD><SPAN DATAFLD="PAGES"></SPAN></TD>
              <TD><SPAN DATAFLD="PRICE"></SPAN></TD>
            </TR>
```

```

        </TABLE>
      </TD>
    </TR>
  </TABLE>
</BODY>
</HTML>

```

В Internet Explorer 5 этот документ будет отображен, как показано на рис. 8.6.

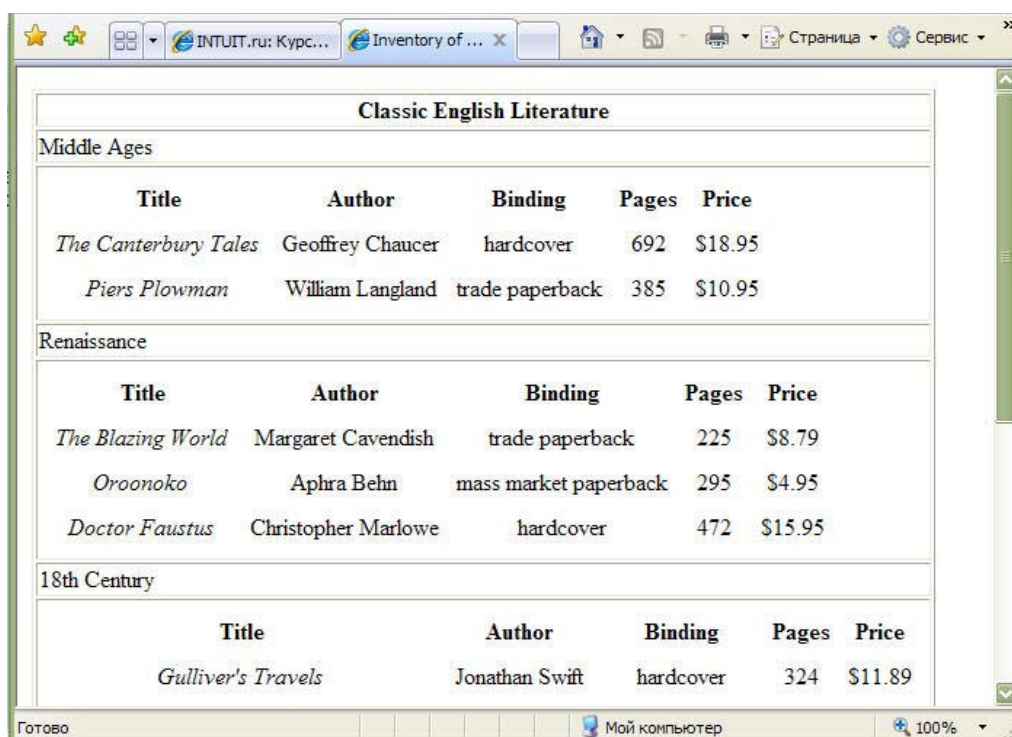


Рис. 8.6. Отображение в Internet Explorer 5 валидного документа

Если данные не отображаются, то в документе имеется ошибка корректности или валидности. Чтобы обнаружить ошибку, воспользуйтесь сценарием проверки на валидность.

8.9. СВЯЗЫВАНИЕ HTML-ЭЛЕМЕНТОВ С XML-АТТРИБУТАМИ

В рассмотренном примере XML-документа ни один из элементов не имел атрибутов. Атрибуты несколько усложняют связывание данных, хотя при этом вы можете сцеплять как элементы, имеющие атрибуты, так и сами атрибуты.

При связывании данных атрибут трактуется как дочерний элемент.

Для элемента record такая трактовка облегчает доступ (или позволяет игнорировать) к атрибуту. Например, следующая запись BOOK содержит атрибут с именем InStock:

```
<BOOK InStock="yes">
  <TITLE>The Adventures of Huckleberry Finn</TITLE>
  <AUTHOR>Mark Twain</AUTHOR>
  <BINDING>mass market paperback</BINDING>
  <PAGES>298</PAGES>
  <PRICE>$5.49</PRICE>
</BOOK>
```

Эта запись трактуется так, как если бы атрибут InStock был полем, принадлежащим BOOK, а значение InStock было бы содержимым этого поля. Таким образом, элемент BOOK воспринимался бы в соответствии со следующей структурой:

```
<BOOK>
  <InStock>yes</InStock>
  <TITLE>The Adventures of Huckleberry Finn</TITLE>
  <AUTHOR>Mark Twain</AUTHOR>
  <BINDING>mass market paperback</BINDING>
  <PAGES>298</PAGES>
  <PRICE>$5.49</PRICE>
</BOOK>
```

Следовательно, вы можете получить доступ к значению атрибута с использованием обычной техники связывания данных. Например, следующий элемент SPAN сцеплен с атрибутом и отображает его значение:

```
<SPAN DATASRC="#dsoInventory" DATAFLD="InStock"></SPAN>
```

(В этом примере предполагается, что XML-документ связан со страницей через фрагмент данных с именем dsoInventory.)

Следует учитывать, что при добавлении атрибута к одному из элементов-полей в XML-документе, например добавлении атрибута к полю AUTHOR, результат будет следующий:

```
<BOOK>
  <TITLE>The Adventures of Huckleberry Finn</TITLE>
  <AUTHOR Born="1835">Mark Twain</AUTHOR>
  <BINDING>mass market paperback</BINDING>
  <PAGES>298</PAGES>
  <PRICE>$5.49</PRICE>
</BOOK>
```

После связывания данных элемент AUTHOR будет интерпретирован следующим образом:

```
<AUTHOR>
  <Born>1835</Born>
  Mark Twain
</AUTHOR>
```

В результате DSO будет хранить элемент как вложенную запись, а не как поле. Следовательно, набор записей превратится в иерархический набор, а не в простой набор записей, и вам придется отображать вложенные записи с использованием вложенной таблицы, как описано в 8.4.3 «Использование вложенных таблиц для отображения иерархической структуры записей»

Чтобы иметь возможность отобразить как символьные данные (Mark Twain), так и атрибут как вложенную запись, вам следует учитывать то обстоятельство, что DSO использует специальное имя TEXT для обращения ко всем символьным данным элемента, не включая при этом значений атрибута. Так, элемент AUTHOR будет интерпретирован следующим образом:

```
<AUTHOR>
  <Born>1835</Born>
  <TEXT>Mark Twain</TEXT>
</AUTHOR>
```

Вы можете использовать имя TEXT в качестве имени поля, чтобы связать ячейку таблицы с символьными данными, содержащимися в записи AUTHOR.

В листинге 8.14 представлена HTML-страница, демонстрирующая все рассмотренные в этой подглаве приемы. Эта страница отображает XML-документ Inventory Valid.xml.

Листинг 8.14. Inventory Attribute.htm

```
<!-- Имя файла: Inventory Attribute.htm -->
<HTML>
<HEAD>
  <TITLE>Book Inventory</TITLE>
</HEAD>
<BODY>
  <XML ID="dsoInventory" SRC="Inventory Valid.xml"></XML>
  <H2>Book Inventory</H2>
  <TABLE DATASRC="#dsoInventory" BORDER="1" CELLPADDING="5">
    <THEAD>
```

```

<TH>Title</TH>
<TH>Author</TH>
<TH>Binding</TH>
<TH>Pages</TH>
<TH>Price</TH>
<TH>In Stock?</TH>
</THEAD>
<TR ALIGN="center">
  <TD>
    <TABLE DATASRC="#dsoInventory" DATAFLD="TITLE">
      <TR>
        <TD><SPAN DATAFLD="$TEXT"></SPAN></TD>
      </TR>
    </TABLE>
  </TD>
  <TD>
    <TABLE DATASRC="#dsoInventory" DATAFLD="AUTHOR">
      <TR>
        <TD><SPAN DATAFLD="$TEXT"></SPAN></TD>
        <TD><SPAN DATAFLD="Born"></SPAN></TD>
      </TR>
    </TABLE>
  </TD>
  <TD><SPAN DATAFLD="BINDING"></SPAN></TD>
  <TD><SPAN DATAFLD="PAGES"></SPAN></TD>
  <TD><SPAN DATAFLD="PRICE"></SPAN></TD>
  <TD><SPAN DATAFLD="InStock"></SPAN></TD>
</TR>
</TABLE>
</BODY>
</HTML>

```

В Internet Explorer 5 этот документ будет отображен, как показано на рис. 8.7.

На этой странице в последнем столбце внешней таблицы отображены значения атрибута InStock записи BOOK путем обычного сцепления его с элементом SPAN:

```
<TD><SPAN DATAFLD="InStock"></SPAN></TD>
```

Поскольку дочерний элемент AUTHOR элемента BOOK содержит атрибут (Born), он интерпретируется как вложенная запись, а не как поле, поэтому страница отображает его с помощью вложенной таблицы:

```

<TD>
  <TABLE DATASRC="#dsoInventory" DATAFLD="AUTHOR">
  <TR>
    <TD><SPAN DATAFLD="(TEXT"></SPAN></TD>
    <TD><SPAN DATAFLD="Born"></SPAN></TD>
  </TR>
  </TABLE>
</TD>

```

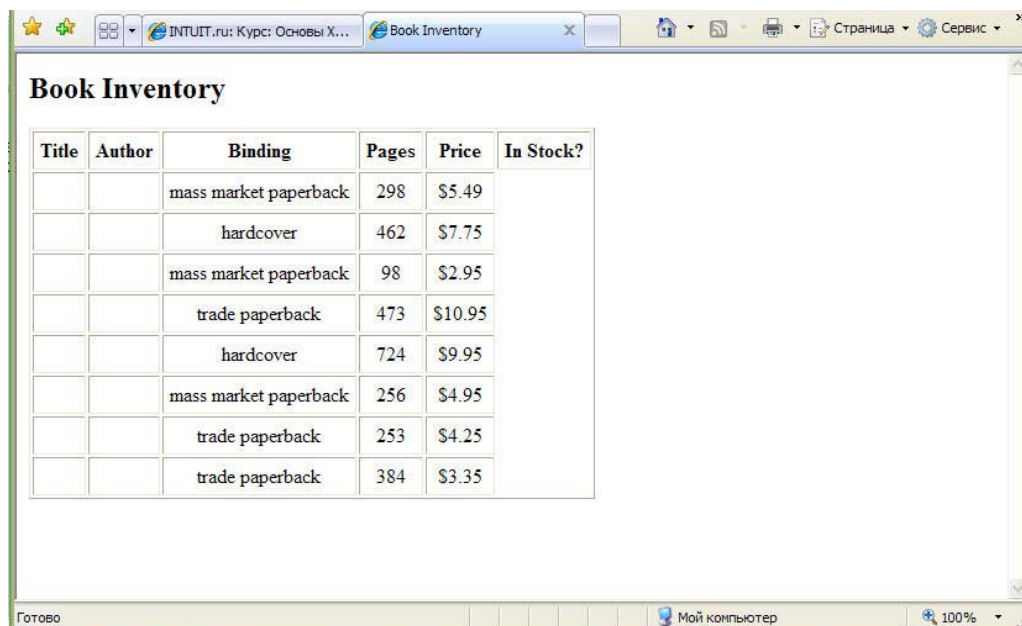


Рис. 8.7. Отображение в Internet Explorer 5 связанных таблиц

Специальное имя TEXT ссылается на весь текст внутри элемента AUTHOR, не включая значения атрибута. Этот текст состоит из имени автора (например, Mark Twain).

Заметим, что поскольку элемент TITLE в BOOK может включать дочерний элемент (SUBTITLE), он также интерпретируется как вложенная запись, а не как поле, и должен также быть отображен с использованием вложенной таблицы:

```

<TD>
  <TABLE DATASRC="#dsoInventory" DATAFLD="TITLE">
  <TR>
    <TD><SPAN DATAFLD="(TEXT"></SPAN></TD>
  </TR>
  </TABLE>
</TD>

```

Здесь TEXT применяется для отображения всех символьных данных записи. (Нельзя установить связь только с текстом заголовка, не включая подзаголовок.)

8.10. ИСПОЛЬЗОВАНИЕ СЦЕНАРИЕВ ДЛЯ DSO

В завершении главы познакомимся с примером более сложного сценария, который использует DSO для работы с соответствующим набором записей XML-документа. В примере использованы методы и свойства объекта DSO recordset для поиска книг в документе Inventory Big.xml. Приемы, используемые для поиска и отображения данных XML, подходят только для XML-документа, организованного как простой набор записей. (Что такое простой набор записей – смотрите в 8.4.1 «Использование одной HTML-таблицы для отображения простого набора записей».)



Совет. С более сложными примерами сценариев вы познакомитесь в главе 9. Сценарии в этой главе используют совершенно иной программный объект (объектную модель XML-документа – Document Object Model), который даст вам возможность работать с любыми типами XML-документов, а не только с документами, структурированными как набор записей.

В листинге 8.15 представлена HTML-страница, содержащая пример сценария.

Листинг 8.15. Inventory Find.htm

```
<!-- Имя файла: Inventory Find.htm -->
<HTML>
<HEAD>
  <TITLE>Book Finder</TITLE>
</HEAD>
<BODY>
  <XML ID="dsoInventory" SRC="Inventory Big.xml"></XML>
  <H2>Find a Book</H2>
  Title text: <INPUT TYPE="TEXT" ID="SearchText">&nbsp;
  <BUTTON ONCLICK='FindBooks()'>Search</BUTTON>
  <HR>
  Results:<P>
  <DIV ID=ResultDiv></DIV>
  <SCRIPT LANGUAGE="JavaScript">
    function FindBooks ()
```



```

{
SearchString = SearchText.value.toUpperCase();
if (SearchString == "")
{
ResultDiv.innerHTML = "&lt;You must enter text into "
+ "'Title text' box.&gt;";
return;
}
dsoInventory.recordset.moveFirst();
ResultHTML = "";
while (!dsoInventory.recordset.EOF)
{
TitleString = dsoInventory.recordset("TITLE").value;
if (TitleString.toUpperCase().indexOf(SearchString)
>=0)
ResultHTML += "<I>"
+ dsoInventory.recordset("TITLE")
+ "</I>, "
+ "<B>"
+ dsoInventory.recordset("AUTHOR")
+ "</B>, "
+ dsoInventory.recordset("BINDING")
+ ", "
+ dsoInventory.recordset("PAGES")
+ " pages, "
+ dsoInventory.recordset("PRICE")
+ "<P>";
dsoInventory.recordset.moveNext();
}
if (ResultHTML == "")
ResultDiv.innerHTML = "&lt;no books found&gt;";
else
ResultDiv.innerHTML = ResultHTML;
}
</SCRIPT>
</BODY>
</HTML>

```

HTML-страница отображает элемент INPUT типа TEXT, который разрешает пользователю ввести одну строку искомого текста:

```
<INPUT TYPE="TEXT" ID="SearchText">
```

Страница также отображает элемент BUTTON (кнопка) с надписью «Search»:

```
<BUTTON ONCLICK='FindBooks()'>Search</BUTTON>
```

Когда пользователь щелкает мышью на кнопке, вызывается функция сценария FindBooks, которая извлекает искомый текст из элемента INPUT и просматривает названия из всех записей BOOK в XML-документе в поисках текста, после чего отображает найденные записи BOOK, содержащие этот текст. В Internet Explorer 5 этот документ будет отображен, как показано на рис. 8.8.

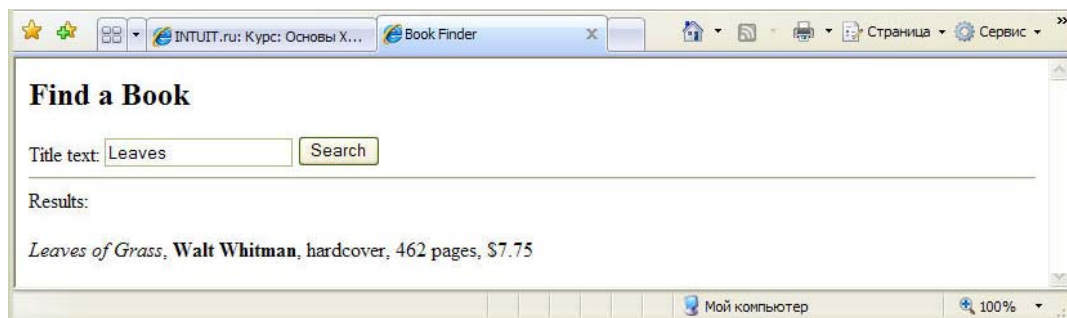


Рис. 8.8. Отображение в Internet Explorer 5 результата применения функции сценария FindBooks

Функция сценария FindBooks содержится в элементе SCRIPT и написана на языке JSCRIPT:

```
<SCRIPT LANGUAGE="JavaScript">
function FindBooks ()
{
    SearchString = SearchText.value.toUpperCase();
    if (SearchString == "")
    {
        ResultDiv.innerHTML = "&lt;You must enter text into"
        + "'Title text' box.&gt;";
        return;
    }
    dsoInventory.recordset.moveFirst();
    ResultHTML = "";
    while (!dsoInventory.recordset.EOF)
    {
        TitleString = dsoInventory.recordset("TITLE").value;
        if (TitleString.toUpperCase().indexOf(SearchString)
            >=0)
            ResultHTML += "<I>"
            + dsoInventory.recordset("TITLE")
            + "</I>, "
            + "<B>"
            + dsoInventory.recordset("AUTHOR")
            + "</B>,"
    }
}
```

```

        + dsoInventory.recordset("BINDING")
        + ", "
        + dsoInventory.recordset("PAGES")
        + " pages, "
        + dsoInventory.recordset("PRICE")
        + "<P>";
    dsoInventory.recordsetmoveNext();
}
if (ResultHTML == "")
    ResultDiv.innerHTML = "&ltno books found&gt;";
else
    ResultDiv.innerHTML = ResultHTML;
}
</SCRIPT>

```

Сначала функция FindBooks получает текст, введенный через элемент INPUT (он имеет атрибут ID SearchText), а затем использует метод toUpperCase JScript для преобразования символов текста в прописные буквы. (Функция FindBooks преобразует текст в прописные буквы, чтобы поиск осуществлялся без учета регистра.)

```
SearchString = SearchText.value.toUpperCase();
```

Если пользователь не ввел текст в поле INPUT, функция отображает сообщение и завершает свою работу:

```

if (SearchString == "")
{
    ResultDiv.innerHTML = "&ltYou must enter text into "
        + "'Title text' box.&gt;";
    return;
}

```

ResultDiv есть идентификатор ID элемента DIV в нижней части страницы, который отображает результаты поиска. Присвоение текста (который может включать HTML-разметку) свойству innerHTML элемента DIV приводит к отображению этого текста (с учетом всей содержащейся в нем HTML-разметки).

Далее функция делает текущей первую запись XML, используя метод recordset.moveFirst, с которым вы познакомились ранее:

```
dsoInventory.recordset.moveFirst();
```

Затем она очищает строковую переменную, используемую для хранения HTML-разметки найденных результатов (ResultHTML):

```
ResultHTML = "";
```

После этого функция FindBooks выполняет цикл просмотра всех записей в XML-документе. Для анализа момента завершения цикла при достижении конца файла используется свойство recordset.EOF, а для перехода к новой записи применяется метод recordset.moveNext:

```
while (!dsoInventory.recordset.EOF)
{
    TitleString = dsoInventory.recordset("TITLE").value;
    if (TitleString.toUpperCase().indexOf(SearchString)
        >=0)
        ResultHTML += "<I>"
            + dsoInventory.recordset("TITLE")
            + "</I>, "
            + "<B>"
            + dsoInventory.recordset("AUTHOR")
            + "</B>,"
            + dsoInventory.recordset("BINDING")
            + ", "
            + dsoInventory.recordset("PAGES")
            + " pages, "
            + dsoInventory.recordset("PRICE")
            + "<P>";
    dsoInventory.recordset.moveNext();
}
```

В начале цикла функция получает значение поля TITLE для текущей записи:

```
TitleString = dsoInventory.recordset("TITLE").value;
```

Выражение справа от знака равенства представляет собой краткую нотацию вызова свойства fields объекта recordset. Полная нотация выглядит следующим образом:

```
TitleString = dsoInventory.recordset.fields("TITLE").value;
```

Свойство fields содержит множество всех полей, принадлежащих текущей записи. Чтобы получить доступ к определенному полю, следует поместить в скобках имя этого поля, и вы получите его содержимое как строку, через свойство value, добавленное в конце выражения.

Далее в цикле используется метод indexOf JScript для анализа, содержит ли название в текущей записи искомый текст. Если искомый текст обнаружен, код внутри оператора if добавляет к строке

ResultHTML текст и HTML-разметку, требуемую для отображения текущей записи:

```
if (TitleString.toUpperCase().indexOf(SearchString)
    >=0)
    ResultHTML += "<I>"
    + dsoInventory.recordset("TITLE")
    + "</I>, "
    + "<B>"
    + dsoInventory.recordset("AUTHOR")
    + "</B>,"
    + dsoInventory.recordset("BINDING")
    + ", "
    + dsoInventory.recordset("PAGES")
    + " pages, "
    + dsoInventory.recordset("PRICE")
    + "<P>";
```

По выходу из цикла функция назначает HTML-разметку, содержащую результаты, свойству innerHTML элемента DIV в разделе BODY документа, который используется для отображения этих результатов (данный элемент DIV имеет идентификатор ResultDiv):

```
if (ResultHTML == "")
    ResultDiv.innerHTML = "&lt;no books found&gt;";
else
    ResultDiv.innerHTML = ResultHTML;
```

Элемент DIV воспринимает HTML-разметку и сразу же отображает результаты.

? ЗАДАНИЯ

1. Отобразите документ по отдельным записям

Откройте новый, пустой текстовый файл в вашем текстовом редакторе и введите содержимое HTML-страницы, представленное в листинге 8.9 (см. на с. 229–230).

Обратите внимание, что страница содержит фрагмент данных, который связан с документом Inventory Big.xml, содержащим 16 записей (см. листинг 8.3 на с. 215–218).

Воспользуйтесь командой Save (Сохранить) вашего текстового редактора, чтобы сохранить страницу на вашем жестком диске, присвоив ей имя файла Inventory Single.htm.

2. Создайте валидный XML-документ для связывания данных

В вашем текстовом редакторе откройте документ Inventory Hierarchy.xml, который вы создали ранее в этой главе.

Непосредственно над элементом Документ (INVENTORY) введите следующее объявление типа:

```
<!DOCTYPE INVENTORY
[
  <!ELEMENT INVENTORY (CATEGORY*)>
  <!ELEMENT CATEGORY (CATNAME, BOOK*)>
  <!ELEMENT CATNAME (#PCDATA)>
  <!ELEMENT BOOK (TITLE, AUTHOR, BINDING, PAGES, PRICE)>
  <!ELEMENT TITLE (#PCDATA)>
  <!ELEMENT AUTHOR (#PCDATA)>
  <!ELEMENT BINDING (#PCDATA)>
  <!ELEMENT PAGES (#PCDATA)>
  <!ELEMENT PRICE (#PCDATA)>
]
>
```

Эти объявления элементов можно объяснить в терминах набора записей и связывания данных следующим образом:

- <!ELEMENT INVENTORY (CATEGORY*)> – документ содержит нуль или несколько записей CATEGORY;
- <!ELEMENT CATEGORY (CATNAME, BOOK*)> – каждая запись CATEGORY содержит одно поле CATNAME, после которого идет нуль или несколько вложенных записей BOOK;
- <!ELEMENT BOOK (TITLE, AUTHOR, BINDING, PAGES, PRICE)> – каждая вложенная запись BOOK содержит по одному из следующих полей в порядке перечисления: TITLE, AUTHOR, BINDING, PAGES и PRICE;
- <!ELEMENT TITLE (#PCDATA)> и все остальные объявления – каждое из полей в записи BOOK содержит только символьные данные.

Чтобы отразить новое имя файла, которое вы присвоите, измените комментарий в начале документа

```
<!-- Имя файла: Inventory Hierarchy.xml -->
```

на

```
<!-- Имя файла: Inventory Hierarchy Valid.xml -->
```

Выберите команду Save As (Сохранить как) вашего текстового редактора, чтобы сохранить копию модифицированного документа, задав в качестве имени файла Inventory Hierarchy Valid.xml.

В листинге 8.12 (см. на с. 238–241) представлен полный XML-документ.

Глава 9. СТРУКТУРА УЗЛОВ XML-ДОКУМЕНТОВ

9.1. СВЯЗЫВАНИЕ XML-ДОКУМЕНТА С HTML-СТРАНИЦЕЙ

Чтобы получить доступ к XML-документу с использованием DOM, вы должны связать XML-документ с HTML-страницей. Самый простой способ – сделать это через фрагмент данных. Напомним, что фрагмент данных создается через HTML-элемент с именем XML. Например, следующий элемент BODY HTML-страницы содержит фрагмент данных, который связывает XML-документ, хранящийся в файле Book.xml:

```
<BODY>  
  <XML ID="dsoBook" SRC="Book.xml"></XML>  
  <!-- Другие элементы отображаемой части страницы. -->  
</BODY>
```

Идентификатор ID, который вы назначаете фрагменту данных, указывает на DSO документа. Вы можете использовать составляющую XMLDocument DSO для доступа к DOM, как показано в следующей строке кода сценария:

```
Document = dsoBook.XMLDocument;
```

Член XMLDocument содержит корневой объект DOM, который известен как узел Document (Document node). Вы можете использовать узел Document для доступа к другим объектам DOM.

Так, создание фрагмента данных на HTML-странице предписывает Internet Explorer 5 создать как DSO (представленный непосредственно через ID фрагмента данных), так и DOM (доступ к которой осуществляется через член XMLDocument DSO).



Совет. Если вы хотите иметь доступ к нескольким XML-документам с HTML-страницы, вы можете поместить фрагмент данных для каждого из них. Кроме того, вы можете включить несколько фрагментов данных для одного XML-документа.

9.2. СТРУКТУРА DOM

В DOM программные объекты, представляющие XML-документ, называются узлами. Когда Internet Explorer 5 обрабатывает связанный XML-документ и сохраняет его в DOM, он создает узел для каждого из основных компонентов XML-документов, таких как элементы, атрибуты и инструкции по обработке.

DOM использует различные типы узлов для представления различных типов компонентов XML. Например, элемент хранится в узле Element, а атрибут – в узле Attribute. В табл. 9.1 представлены наиболее важные типы узлов.

Таблица 9.1

Типы узлов компонентов XML

Тип узла DOM	Компоненты XML-документа, представляемые узлом	Имя узла (свойство nodeName объекта)	Значение узла (свойство nodeValue объекта)
1	2	3	4
Document	Корневой узел иерархии документа	#document	null
Element	Элемент	Имя типа элемента (например, BOOK)	null (любые символьные данные, которые находятся в одном или нескольких дочерних узлах Text)
Text	Текст, принадлежащий элементу, атрибуту или примитиву, которые представлены родителем этого узла	#text	Текст родительского XML-компонента
Attribute	Атрибут (а также другие пары «имя – значение», такие как имя и значение в инструкции по обработке)	Имя атрибута (например, Binding)	Значение атрибута (например, hardcover)
Processing-Instruction	Инструкция по обработке (объявление XML или пользовательская инструкция по обработке)	Предназначение инструкции по обработке (например, xml)	Полное содержимое инструкции по обработке, за исключением предназначения
Comment	Комментарий	#comment	Весь текст внутри ограничителей комментария

1	2	3	4
CDATASection	Раздел CDATA	#cdata-section	Содержимое раздела CDATA
DocumentType	Объявление типа документа	Имя корневого элемента, содержащееся в объявлении DOCTYPE (например, INVENTORY)	null
Entity	Объявление примитива в DTD	Имя примитива (например, image)	null (значение примитива содержится в дочернем узле Text)
Notation	Объявление нотации в DTD	Имя нотации (например, BMP)	null (системный литерал нотации содержится в дочернем узле Attribute с именем SYSTEM)

Вы можете получить каждое из имен узлов из свойства узла `nodeName`. Имена, начинающиеся с символа `#`, представляют компоненты XML, не поименованные в документе. Другие имена узлов получаются из имен, присвоенных соответствующим компонентам в XML-документе.

Вы можете получить каждое из значений узла (перечисленные в последнем столбце) из свойства узла `nodeValue`. Если компонент XML имеет соответствующее значение (например, атрибут), это значение будет храниться в значении узла. Если компонент XML не имеет значения (например, элемент), DOM устанавливает в качестве значения узла `null`. Подробнее о большинстве типов узлов, перечисленных в табл. 9.1, вы узнаете далее в этой главе.

DOM организует узлы XML-документа в виде древовидной иерархической структуры, которая отражает иерархическую структуру самого документа. При этом создается единственный узел `Document`, который представляет весь XML-документ и служит корневым элементом в этой иерархии. Заметим, что логическая иерархическая структура элементов XML, в которой элемент Документ является корневым, — это лишь одна из ветвей иерархической структуры узлов DOM, которые представляют весь документ.

Возьмем, например, XML-документ из листинга 9.1. Этот документ состоит из объявления XML, комментария и корневого элемента, который включает дочерние элементы, а также атрибуты.

На рис. 9.1 показана иерархическая организация узлов, которые создает DOM для представления документа. Для каждого компонента рассматриваемого документа на схеме указан тип узла, используемого для представления компонента (например, Document, Comment и Element), а также имя узла (оно указано в скобках – например, #document, #comment и INVENTORY).

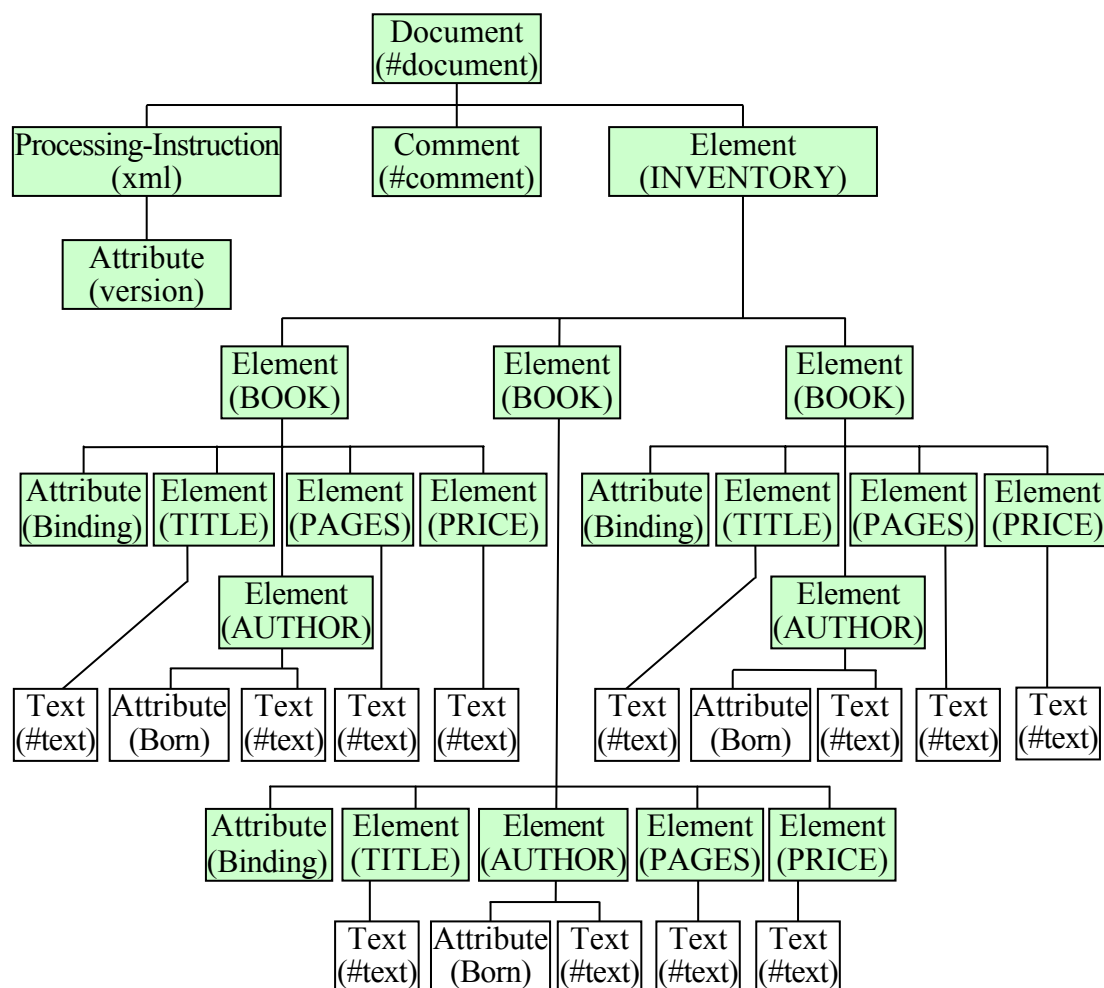


Рис. 9.1. Иерархическая организация узлов

Листинг 9.1. Inventory Dom.xml

```

<?xml version="1.0"?>
<!-- Имя файла: Inventory Dom.xml -->
<INVENTORY>
    <BOOK Binding="mass market paperback">
        <TITLE>The Adventures of Huckleberry Finn</TITLE>
        <AUTHOR Born="1835">Mark Twain</AUTHOR>
    </BOOK>
    <BOOK>
        <TITLE>The Adventures of Tom Sawyer</TITLE>
        <AUTHOR Born="1835">Mark Twain</AUTHOR>
    </BOOK>
    <BOOK>
        <TITLE>The Adventures of Huckleberry Finn</TITLE>
        <AUTHOR Born="1835">Mark Twain</AUTHOR>
    </BOOK>
</INVENTORY>

```

```

        <PAGES>298</PAGES>
        <PRICE>$5.49</PRICE>
    </BOOK>
    <BOOK Binding="trade paperback">
        <TITLE>The Marble Faun</TITLE>
        <AUTHOR Born="1804">Nathaniel Hawthorne</AUTHOR>
        <PAGES>473</PAGES>
        <PRICE>$10.95</PRICE>
    </BOOK>
    <BOOK Binding="hardcover">
        <TITLE>Moby-Dick</TITLE>
        <AUTHOR Born="1819">Herman Melville</AUTHOR>
        <PAGES>724</PAGES>
        <PRICE>$9.95</PRICE>
    </BOOK>
</INVENTORY>

```

Каждый узел, как и программный объект, имеет свойства и методы, которые позволяют вам осуществлять доступ, отображать, обрабатывать и получать информацию о соответствующем компоненте XML. Например, свойства `nodeName` и `nodeValue` (см. табл. 9.1 на с. 256–257) дают имя компонента и его значение.

Все типы узлов используют общий набор свойств и методов. Эти свойства и методы разработаны для работы с узлами вообще. В табл. 9.2 представлены некоторые наиболее полезные свойства.

Таблица 9.2

Полезные свойства, поддерживаемые всеми типами узлов

Свойство	Описание	Пример
1	2	3
<code>Attributes</code>	Множество <code>NamedNodeMap</code> всех дочерних узлов-атрибутов данного узла	<code>AttributeNode = Element.attributes.getNamedItem("Binding");</code>
<code>childNodes</code>	Множество <code>NodeList</code> всех дочерних узлов, не являющихся атрибутами данного узла	<code>FirstNode = Element.childNodes(0);</code>
<code>dataType</code>	Тип данных этого узла применительно только к определенным типам узлов <code>Attribute</code>	<code>AttributeType = Attribute.dataType;</code>

1	2	3
firstChild	Первый дочерний узел данного узла, не являющийся атрибутом	FirstChildNode = Element.firstChild;
lastChild	Последний дочерний узел данного узла, не являющийся атрибутом	LastChildNode = Element.lastChild;
nextSibling	Следующий узел на том же уровне данного узла	NextElement = Element.nextSibling;
nodeName	Имя данного узла	ElementName = Element.nodeName;
nodeType	Цифровой код, указывающий на тип данного узла	NodeTypeCode = Node.nodeType;
nodeTypeString	Строка, содержащая тип данного узла, строчными буквами (например, «element» или «attribute»)	NodeTypeString = Node.nodeTypeString;
nodeValue	Значение данного узла (или null, если он не содержит значения)	AttributeValue = Attribute.nodeValue;
ownerDocument	Корневой узел Document документа, содержащего данный узел	Document = Node.ownerDocument;
parentNode	Узел, для которого данный узел является дочерним (не действует для узла Attribute)	ParentElement = Element.parentNode;
previousSibling	Предыдущий узел на том же уровне данного узла	PreviousElement = Element.previousSibling;
text	Все текстовое содержимое данного узла и все подчиненные узлы Element	AllCharacterData = Element.text;
xml	Все содержимое XML данного узла и все его подчиненные узлы	XMLContent = Element.xml;



Совет. Если вы хотите познакомиться со всеми свойствами, методами и событиями, принадлежащими различным типам объектов-узлов, обратитесь к следующей Web-странице, предоставленной MSDN:

http://msdn.microsoft.com/xml/reference/scriptref/xml_dom_Objects.asp.
Заметьте, что в табл. 9.1 (см. на с. 256–257) каждому имени узла предшествует префикс XMLDOM, например XMLDOMDocument, XMLDOMElement и XMLDOMText. (Это имена программных интерфейсов для каждого типа узла.) Обратите внимание также, что общие свойства и методы узлов представлены под именем XMLDOMNode.

Помимо общих свойств и методов, каждому типу узла присущи дополнительные свойства и методы, разработанные для работы с определенным XML-компонентом, который представляет узел. Например, узел `Document` имеет свойство `parseError`, которое содержит информацию о любой ошибке, возникающей в процессе обработки документа. Данное свойство характерно только узлу `Document`. Далее в этой главе вы познакомитесь с табл. 9.3 (см. на с. 265–266), в которой представлены наиболее полезные свойства и методы для некоторых типов узлов.



Совет. Свойство будет иметь значение `null`, если данное свойство не применимо к определенному узлу. Например, если узел представляет XML-компонент, который не имеет атрибутов (например, узел `Document` или `Comment`), его свойство `attributes` будет иметь значение `null`. Если узел представляет XML-компонент, который не имеет типа данных (тип данных имеют только определенные атрибуты), его свойство `dataType` будет иметь значение `null`. Если узел не имеет дочернего узла, не являющегося атрибутом, его свойство `firstChild` будет иметь значение `null`. Если узел относится к типу, который не имеет значений (например, узел `Document` или `Element`), его свойство `nodeValue` также будет иметь значение `null`.

Обратите внимание, что в табл. 9.2 каждый узел обладает набором свойств, которые позволяют вам перемещаться в иерархии узла, т. е. получать доступ к другим узлам от текущего узла. Например, рассмотрим документ из листинга 9.1. Если переменная `Document` содержит корневой узел `Document`, следующий код приведет к отображению содержимого комментария, расположенного в начале документа (этот комментарий DOM хранит как второй дочерний узел узла `Document`):

```
alert (Document.childNodes(1).nodeValue);
```

Эта строка вызовет отображение сообщения, содержащего текст «Имя файла: Inventory Dom.xml».

В предыдущей подглаве вы узнали, как осуществлять доступ к корневому узлу `Document` через член `XMLDocument` DSO, который получается из фрагмента данных XML. Узел `Document` является шлюзом к XML-документу. Вы можете использовать его для доступа к другим узлам. В последующих подглавах вы получите представление об особых способах доступа к узлам.

9.3. ДОСТУП И ОТОБРАЖЕНИЕ ЭЛЕМЕНТОВ XML-ДОКУМЕНТА

В этой подглаве вы познакомитесь с основными приемами использования HTML-страницы и DOM для отображения элементов XML-документа.

Листинг 9.2. Book.xml

```
<?xml version="1.0"?>
<!-- Имя файла: Book.xml -->
<BOOK>
  <TITLE>The Adventures of Huckleberry Finn</TITLE>
  <AUTHOR>Mark Twain</AUTHOR>
  <BINDING>mass market paperback</BINDING>
  <PAGES>298</PAGES>
  <PRICE>$5.49</PRICE>
</BOOK>
```

Листинг 9.3. DomDemo Fixed.htm

```
!-- Имя файла: DomDemo Fixed.htm -->
<HTML>
<HEAD>
  <TITLE>Book Description</TITLE>
  <SCRIPT LANGUAGE = "JavaScript" FOR="window" EVENT="ONLOAD">
    Document = dsoBook.XMLDocument;
    title.innerText =
      Document.documentElement.childNodes(0).text;
    author.innerText =
      Document.documentElement.childNodes(1).text;
    binding.innerText =
      Document.documentElement.childNodes(2).text;
    pages.innerText =
      Document.documentElement.childNodes(3).text;
    price.innerText =
      Document.documentElement.childNodes(4).text;
  </SCRIPT>
</HEAD>
<BODY>
  <XML ID="dsoBook" SRC="Book.xml"></XML>
  <H2>Book Description</H2>
  <SPAN STYLE="font-style:italic">Title: </SPAN>
  <SPAN ID="title" STYLE="font-weight:bold"></SPAN>
  <BR>
  <SPAN STYLE="font-style:italic">Author: </SPAN>
  <SPAN ID="author"></SPAN>
  <BR>
```

```

<SPAN STYLE="font-style:italic">Binding: </SPAN>
<SPAN ID="binding"></SPAN>
<BR>
<SPAN STYLE="font-style:italic">Number of pages: </SPAN>
<SPAN ID="pages"></SPAN>
<BR>
<SPAN STYLE="font-style:italic">Price: </SPAN>
<SPAN ID="price"></SPAN>
</BODY>
</HTML>

```

Листинг 9.2 содержит простой XML-документ, который описывает одну книгу. Его корневой элемент – BOOK состоит из пяти дочерних элементов (TITLE, AUTHOR, BINDING, PAGES и PRICE), каждый из которых содержит символьные данные, описывающие характеристики книги.

Листинг 9.3 представляет HTML-страницу, которая отображает содержимое каждого из дочерних элементов в XML-документе (рис. 9.2).

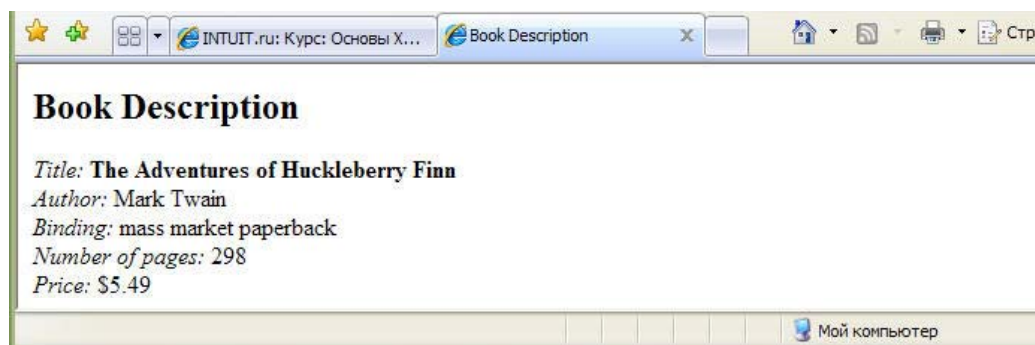


Рис. 9.2. Отображение в Internet Explorer 5 содержимого каждого из дочерних элементов в XML-документе

XML-документ связан со страницей через следующий фрагмент данных:

```
<XML ID="dsoBook" SRC="Book.xml"></XML>
```

Страница отображает XML-документ посредством следующего блока кода сценария, который содержится в элементе HEAD страницы:

```

<SCRIPT LANGUAGE="JavaScript" FOR="window" EVENT="ONLOAD">
  Document = dsoBook.XMLDocument;
  title.innerText =
    Document.documentElement.childNodes(0).text;
  author.innerText =

```

```

        Document.documentElement.childNodes(1).text;
binding.innerText =
        Document.documentElement.childNodes(2).text;
pages.innerText =
        Document.documentElement.childNodes(3).text;
price.innerText =
        Document.documentElement.childNodes(4).text;
</SCRIPT>

```

Установки атрибутов FOR= «window» и EVENT= «ONLOAD» предписывают браузеру выполнять код из элемента SCRIPT при первом открытии окна страницы до того, как будет отображено содержимое страницы.

Сценарий первым делом получает узел Document, который представляет весь документ и формирует корневой элемент иерархии узлов DOM. Он делает это через член XMLDocument DSO, как было описано ранее в этой главе:

```
Document = dsoBook.XMLDocument;
```

Далее сценарий получает доступ и отображает символьные данные, содержащиеся в каждом из дочерних элементов корневого элемента (TITLE, AUTHOR, BINDING, PAGES и PRICE). Например, он отображает содержимое первого дочернего элемента (TITLE) следующим образом:

```
title.innerText = Document.documentElement.childNodes(0).text;
```

Вот пояснение выражения, стоящего справа от знака равенства.

Document содержит узел Document в основании (корне) иерархии узлов DOM.

documentElement – это свойство узла Document. Оно содержит узел Element, представляющий корневой элемент XML-документа (в нашем примере – BOOK).

⇒ *Примечание.* Свойство documentElement является одним из специфических для конкретного узла свойств, предоставляемых узлом типа Document. В табл. 9.3 приведены другие полезные свойства, относящиеся к узлу Document. Имейте в виду, что для узла Document вы можете использовать и общие свойства узлов (см. табл. 9.2 на с. 259–260).

childNodes является свойством узла Element для корневого элемента. Оно содержит множество всех дочерних узлов корневого

узла `Element`, не являющихся атрибутами. В нашем примере оно содержит узлы `Element` для пяти дочерних XML-элементов: `TITLE`, `AUTHOR`, `BINDING`, `PAGES` и `PRICE`. Выражение `childNodes(0)` ссылается на первый из этих дочерних узлов (а именно, на элемент `TITLE`).

⇒ *Примечание.* В рассматриваемом примере страницы (см. листинг 9.3 на с. 262–263) вы можете использовать выражение `Document.childNodes(2)`, чтобы получить доступ к узлу корневого элемента. Однако преимущество использования свойства `documentElement` узла `Document` заключается в том, что его значение не зависит от положения корневого элемента внутри XML-документа. Например, если бы вы удалили комментарий в начале документа либо если бы добавили объявление типа документа, выражение `Document.childNodes(2)` больше не представляло бы корневой элемент.

`text` является свойством узла, возвращаемого выражением `childNodes(0)`. Оно предоставляет весь текст, содержащийся в этом узле, а также текст, принадлежащий любому подчиненному узлу `Element`. В нашем примере `TITLE` не имеет подчиненных элементов, поэтому свойство `text` содержит только собственно текст элемента `TITLE`, «The Adventures of Huckleberry Finn».

⇒ *Примечание.* Свойства `childNodes` и `text` относятся к общим свойствам узлов.

Таблица 9.3

Свойства узла `Document`

Свойство	Описание	Пример
1	2	3
<code>doctype</code>	Узел <code>DocumentType</code> , представляющий объявление типа документа	<code>DocumentType = Document.doctype;</code>
<code>documentElement</code>	Узел <code>Element</code> , представляющий корневой элемент	<code>RootElement = Document.documentElement;</code>
<code>ondataavailable</code>	Если вы присвоите этому свойству имя функции, которую вы написали, функция будет вызываться в момент доступности данных XML	<code>Document.ondataavailable = MyDataAvailableHandler;</code>

1	2	3
onreadystatechange	Если вы присвоите этому свойству имя функции, которую вы написали, функция будет вызываться всякий раз, когда изменяется свойство readyState узла Document	Document.onreadystatechange = MyReadyStateHandler;
parseError	Объект, который содержит информацию о любых ошибках, возникающих в процессе обработки документа	ErrorCode = Document.parseError.errorCode;
readyState	Текущий статус загрузки и обработки XML-документа. Может принимать одно из следующих числовых значений: 0 – не инициализирован; 1 – загружается; 2 – загружен; 3 – интерактивный режим; 4 – завершение	if (Document.readyState = 4) /* обработка данных... */
url	URL XML-документа	URL = Document.url;
getElementsByTagName (type-name)	Возвращает множество NodeList всех элементов в документе, которые имеют заданное имя типа. Если указано «*», возвращает все элементы	AuthorElementCollection = Document.getElementsByTagName ("AUTHOR");
nodeFromID (id-value)	Возвращает узел, представляющий элемент, чей атрибут типа ID имеет указанное значение	Element = Document.nodeFromID ("S021");

Символьные данные элемента TITLE, которые получены из выражения справа от знака равенства («The Adventures of Huckleberry Finn»), присваиваются свойству innerText HTML-элемента SPAN, имеющему идентификатор title:

```
title.innerText = Document.documentElement.childNodes(0).text;
```

Этот элемент SPAN определен внутри элемента BODY HTML-страницы следующим образом:

```
<SPAN ID="title" STYLE="font-weight:bold"></SPAN>
```

Присвоение символьных данных свойству `innerText` элемента `SPAN` приводит к тому, что элемент `SPAN` отображает текст с использованием формата, определенного в его начальном теге (`font-weight:bold`).



Совет. В Dynamic HTML (DHTML) каждый HTML-элемент имеет набор свойств. Свойство `innerText` устанавливает или получает текстовое содержимое элемента.

9.4. ИСПОЛЬЗОВАНИЕ ОБЪЕКТА NODELIST

Свойство `childNodes` узла содержит набор дочерних узлов текущего узла, не являющихся атрибутами. (Доступ к дочерним узлам-атрибутам осуществляется через свойство `attribute` узла.) Определенный тип набора, который содержит свойство `childNodes`, носит название объекта `NodeList`.

В табл. 9.4 приведены свойство и несколько полезных методов, предоставляемых объектом `NodeList`.

Таблица 9.4

Свойство и методы, поддерживаемые групповым объектом `NodeList`

Свойство	Описание	Пример
<code>length</code>	Количество узлов, содержащихся в наборе	<code>NodeCount = Element.childNodes.length;</code>
Метод	Описание	Пример
<code>item</code> (индекс, отсчитываемый с нуля) (метод по умолчанию)	Возвращает узел в соответствии с заданным вами индексом, при этом нуль соответствует первому узлу	<code>SecondChild = Element.childNodes.item(1);</code> или <code>SecondChild = Element.childNodes(1);</code>
<code>reset()</code>	Устанавливает внутренний указатель на позицию перед первым узлом в наборе, чтобы последующий вызов <code>nextNode</code> возвращал первый узел	<code>Element.childNodes.reset();</code>
<code>nextNode()</code>	Возвращает следующий узел в наборе в соответствии с позицией внутреннего указателя	<code>Element.childNodes.reset();</code> <code>FirstNode = Element.childNodes.nextNode();</code>

Чтобы извлечь определенный дочерний узел из объекта `NodeList`, вы можете обратиться к его методу `item`, указав при этом индекс дочернего узла, который вы хотите получить (индексы отсчитываются с нуля). Например, обращение к следующему методу позволяет получить первый дочерний узел, принадлежащий узлу `Element`:

```
FirstNode = Element.childNodes.item(0);
```

Однако, поскольку `item` является методом по умолчанию объекта `NodeList`, вы можете опустить его, как это делалось в предыдущих примерах в этой главе:

```
FirstNode = Element.childNodes(0);
```

9.5. ИЗВЛЕЧЕНИЕ СИМВОЛЬНЫХ ДАННЫХ ЭЛЕМЕНТА

В сценарии, представленном в листинге 9.3 (см. на с. 262–263), свойство `text` каждого из дочерних элементов (`TITLE`, `AUTHOR`, `BINDING`, `PAGES` и `PRICE`) используется для получения символьных данных элемента. Например, следующий оператор применяется для извлечения символьных данных элемента `TITLE`:

```
title.innerText = Document.documentElement.childNodes(0).text;
```

Свойство `text` показывает содержимое элемента в текущем узле плюс текстовое содержимое любого подчиненного элемента. Оно хорошо подходит для извлечения символьных данных элемента в том случае, если элемент не имеет дочерних элементов (например, элемент `TITLE`). Однако если элемент содержит один или более дочерних элементов помимо символьных данных, как в приведенном ниже примере, свойство `text` возвращает весь текст:

```
<TITLE>Moby-Dick
  <SUBTITLE>Or, the Whale</SUBTITLE>
</TITLE>
```

Чтобы получить только символьные данные элемента `TITLE`, вам потребуется осуществить доступ к дочернему узлу `Text`.

Как видно из табл. 9.1 (см. на с. 256–257), свойство `nodeValue` узла `Element` имеет значение `null`. Если элемент содержит сим-

вольные данные, то они хранятся в дочернем узле Text, и вы можете получить их через свойство `nodeValue` узла Text. Например, чтобы получить данные элемента TITLE («Moby-Dick») из предыдущего примера, без символьных данных, которые принадлежат элементу SUBTITLE, используем следующее выражение:

```
Element.firstChild.nodeValue
```

Если символьные данные элемента смешаны с дочерними элементами, комментариями или инструкциями по обработке, каждый отдельный блок символьных данных представляется собственным дочерним узлом Text. Например, приведенный ниже элемент ITEM имеет три дочерних узла, расположенных в следующем порядке: узел TEXT, представляющий первый блок символьных данных; узел ELEMENT, представляющий дочерний элемент SUB-ITEM; еще один узел TEXT, представляющий второй блок символьных данных:

```
<ITEM>
  блок символьных данных № 1 /* узел TEXT */
  <SUB-ITEM>текст подчиненного элемента</SUB-ITEM>
  блок символьных данных № 2 /* узел TEXT */
</ITEM>
```

Свойство и метод, предоставляемые узлом Text, приведены в табл. 9.5.

Таблица 9.5

Полезные свойство и метод, поддерживаемые узлами Text

Свойство	Описание	Пример
length	Количество символов в тексте узла	CharacterCount = Text.length;
Метод	Описание	Пример
substringData (char-offset, num-chars)	Возвращает строку, содержащую заданное число символов из текстового содержимого узла, начиная с указанной параметром смещения позиции	Substring = Text.substringData(2, 3); (Возвращает третий, четвертый и пятый символы из содержимого элемента Text.)

9.6. ОТОБРАЖЕНИЕ ПЕРЕМЕННОГО ЧИСЛА XML-ЭЛЕМЕНТОВ

Итак, вы теперь знаете, как отобразить XML-документ, имеющий известное число элементов. Если документ имеет неизвестное число элементов, использование DOM для отображения документа несколько усложняется.

Например, для XML-документа типа `Inventory.xml` или `Inventory Big.xml` вы обычно не знаете заранее, сколько элементов `BOOK` содержит документ.

В листинге 9.4 представлена HTML-страница, которая использует DOM для отображения документа `Inventory.xml` вне зависимости от того, сколько элементов `BOOK` в нем содержится (рис. 9.3).

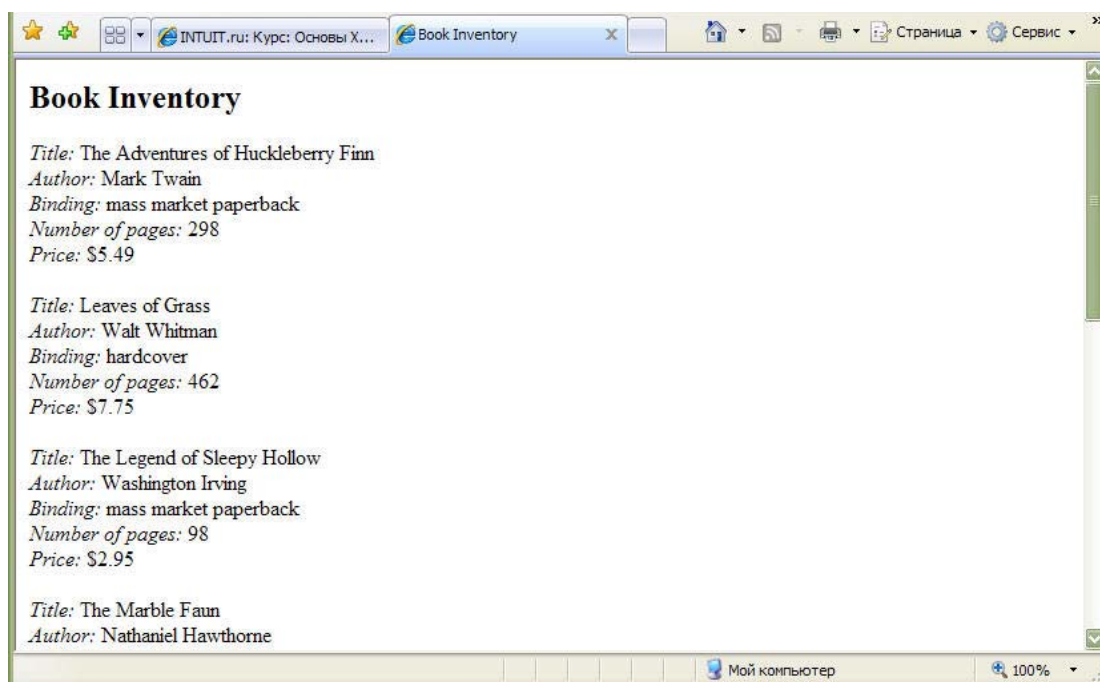


Рис. 9.3. Отображение в Internet Explorer 5 файла `Inventory.xml`

Листинг 9.4. DomDemo Variable.htm

```
<!-- Имя файла: DomDemo Variable.htm -->
<HTML>
<HEAD>
  <TITLE>Book Inventory</TITLE>
  <SCRIPT LANGUAGE="JavaScript" FOR="window" EVENT="ONLOAD">
    HTMLCode = "";
```

```

Document = dsoInventory.XMLDocument;
for (i=0;
    i < Document.documentElement.childNodes.length;
    i++)
{HTMLCode +=
"<SPAN STYLE='font-style:italic'>Title: </SPAN>"
+ Document.documentElement.childNodes(i).
    childNodes(0).text
+ "<BR>"
+ "<SPAN STYLE='font-style:italic'>Author: </SPAN>"
+ Document.documentElement.childNodes(i).
    childNodes(1).text
+ "<BR>"
+ "<SPAN STYLE='font-style:italic'>Binding: </SPAN>"
+ Document.documentElement.childNodes(i).
    childNodes(2).text
+ "<BR>"
+ "<SPAN STYLE='font-style:italic'>Number of pages:"
+ "</SPAN>"
+ Document.documentElement.childNodes(i).
    childNodes(3).text
+ "<BR>"
+ "<SPAN STYLE='font-style:italic'>Price: </SPAN>"
+ Document.documentElement.childNodes(i).
    childNodes(4).text
+ "<P>";
}
DisplayDIV.innerHTML=HTMLCode;
</SCRIPT>
</HEAD>
<BODY>
  <XML ID="dsoInventory" SRC="Inventory.xml"></XML>
  <H2>Book Inventory</H2>
  <DIV ID="DisplayDIV"></DIV>
</BODY>
</HTML>

```

Сценарий в рассматриваемом примере использует свойство `length` для определения количества элементов BOOK внутри корневого элемента. (Свойство `length` является членом группового объекта `NodeList`, предоставляемого свойством `childNodes` узла корневого элемента (см. табл. 9.4 на с. 267).) В сценарии имеется цикл `for`, который выполняется для каждого элемента BOOK и включает код для отображения каждого из этих элементов:

```

for (i=0;
    i < Document.documentElement.childNodes.length;
    i++)
{
    /* код для отображения элемента BOOK... */
}

```

Поскольку количество элементов BOOK неизвестно, страница не может использовать фиксированный набор элементов SPAN в разделе BODY для отображения данных (как это делалось в предыдущем примере из листинга 9.3). Вместо этого для каждого элемента BOOK сценарий динамически генерирует весь блок HTML-разметки, необходимый для отображения элемента:

```

for (i=0;
    i < Document.documentElement.childNodes.length;
    i++)
{
    HTMLCode +=
    "<SPAN STYLE='font-style:italic'>Title: </SPAN>"
    + Document.documentElement.childNodes(i).childNodes(0).text
    + "<BR>"
    + "<SPAN STYLE='font-style:italic'>Author: </SPAN>"
    + Document.documentElement.childNodes(i).childNodes(1).text
    + "<BR>"
    + "<SPAN STYLE='font-style:italic'>Binding: </SPAN>"
    + Document.documentElement.childNodes(i).childNodes(2).text
    + "<BR>"
    + "<SPAN STYLE='font-style:italic'>Number of pages:"
    + "</SPAN>"
    + Document.documentElement.childNodes(i).childNodes(3).text
    + "<BR>"
    + "<SPAN STYLE='font-style:italic'>Price: </SPAN>"
    + Document.documentElement.childNodes(i).childNodes(4).text
    + "<P>";
}

```

Сценарий хранит все эти блоки HTML-разметки в переменном HTMLCode. После цикла for, когда все блоки сгенерированы и загружены в HTMLCode, сценарий присваивает HTML-разметку свойству innerHTML элемента DIV раздела BODY страницы (этот элемент имеет ID DisplayDIV):

```

DisplayDIV.innerHTML=HTMLCode;

```


Элемент DIV затем сразу же получает HTML-разметку и отображает результаты. Чтобы убедиться, что страница работала независимо от количества элементов BOOK, которое содержится в XML-документе, вы можете отредактировать фрагмент данных на этой странице, чтобы он отображал документ Inventory Big.xml, который содержит в 2 раза больше элементов BOOK, чем документ Inventory.xml:

```
<XML ID="dsoInventory" SRC="Inventory Big.xml"></XML>
```

9.7. ИСПОЛЬЗОВАНИЕ ДРУГИХ СПОСОБОВ ДОСТУПА К ЭЛЕМЕНТАМ

В рассмотренных примерах сценариев доступ к узлам Element в иерархической структуре осуществлялся с использованием свойств childNodes или firstChild узла. При этом происходил переход от одного узла к другому. Аналогичным образом вы можете применять свойства узла lastChild, previousSibling, nextSibling и parentNode (см. табл. 9.2 на с. 259–260).

⇒ *Примечание.* Свойства childNodes, firstChild и lastChild дают возможность доступа только к дочерним узлам, которые не являются атрибутами, в то время как свойства previousSibling и nextSibling могут быть использованы для доступа к вершинному узлу любого типа.

Другим способом доступа к XML-элементам является использование свойства getElementsByTagName, которое позволяет извлечь все элементы, имеющие определенное имя типа (например, TITLE). Этот метод может применяться для узла Document (см. табл. 9.3 на с. 265–266), а также для узла Element (см. табл. 9.6). Если вы обращаетесь к методу для узла Document, он возвращает набор узлов Element для всех элементов в документе, обладающих заданным именем типа. Например, следующий оператор позволяет получить группу узлов для всех элементов в документе, обладающих именем BOOK:

```
NodeList = Document.getElementsByTagName("BOOK");
```

Если вы обращаетесь к методу getElementsByTagName для узла Element, как в следующем примере, он возвращает набор узлов

для всех соответствующих элементов, которые являются подчиненными для узла Element:

```
NodeList = Element.getElementsByTagName("AUTHOR");
```



Совет. Если в качестве параметра метода `getElementsByTagName` вы укажете «*», то метод вернет набор узлов для всех элементов.

Таблица 9.6

Полезные методы, поддерживаемые узлами Element

Метод	Описание	Пример
<code>getAttribute</code> (имя-атрибута)	Возвращает значение атрибута элемента с заданным именем	<code>AttValue = Element.getAttribute ("InStock");</code>
<code>getAttributeNode</code> (имя-атрибута)	Возвращает узел <code>Attribute</code> , представляющий атрибут элемента с заданным именем	<code>Attribute = Element.getAttributeNode ("InStock");</code>
<code>getElementsByTagName</code> (имя-типа)	Возвращает набор <code>NodeList</code> узлов <code>Element</code> для всех подчиненных элементов элемента с заданным именем. Если указано «*», возвращает узлы для всех подчиненных элементов	<code>AuthorElementCollection = Element.getElementsByTagName ("AUTHOR");</code>

Метод `getElementsByTagName` предоставляет узлы `Element` в виде группового объекта `NodeList`. Доступ к отдельным узлам осуществляется с помощью действий, описанных в 9.4 «Использование объекта `NodeList`». Например, следующий код отображает (в окне предупреждения) текстовое содержимое всех узлов `Element` в объекте `NodeList`, возвращенное методом `getElementsByTagName`:

```
for (i=0; i < NodeList.length; ++i)
    alert (NodeList(i).text);
```

HTML-страница из листинга 9.5 демонстрирует использование метода `getElementsByTagName` для узла `Document`. Страница отображает поле ввода `INPUT` типа `TEXT`, которое дает вам возможность ввести имя элемента. Когда вы щелкаете мышью на кнопке `ShowElements`, вызывается функция `ShowElements` сценария,

которая использует метод `getElementsByTagName` для узла `Document`, чтобы найти и отобразить XML-разметку для всех элементов в документе, которые носят введенное вами имя элемента (если они имеются).

Заметим, что сценарий использует свойство `xml` каждого из узлов `Element` для отображения содержимого XML-разметки элемента. Страница изначально связана с документом `Inventory.xml`, хотя вы можете отредактировать фрагмент данных, чтобы отобразить элементы из другого XML-документа.

На рис. 9.4 показано, как Internet Explorer 5 отобразит страницу после того, как вы ввели в поле ввода INPUT имя `AUTHOR` и щелкнули мышью на кнопке `ShowElements`.

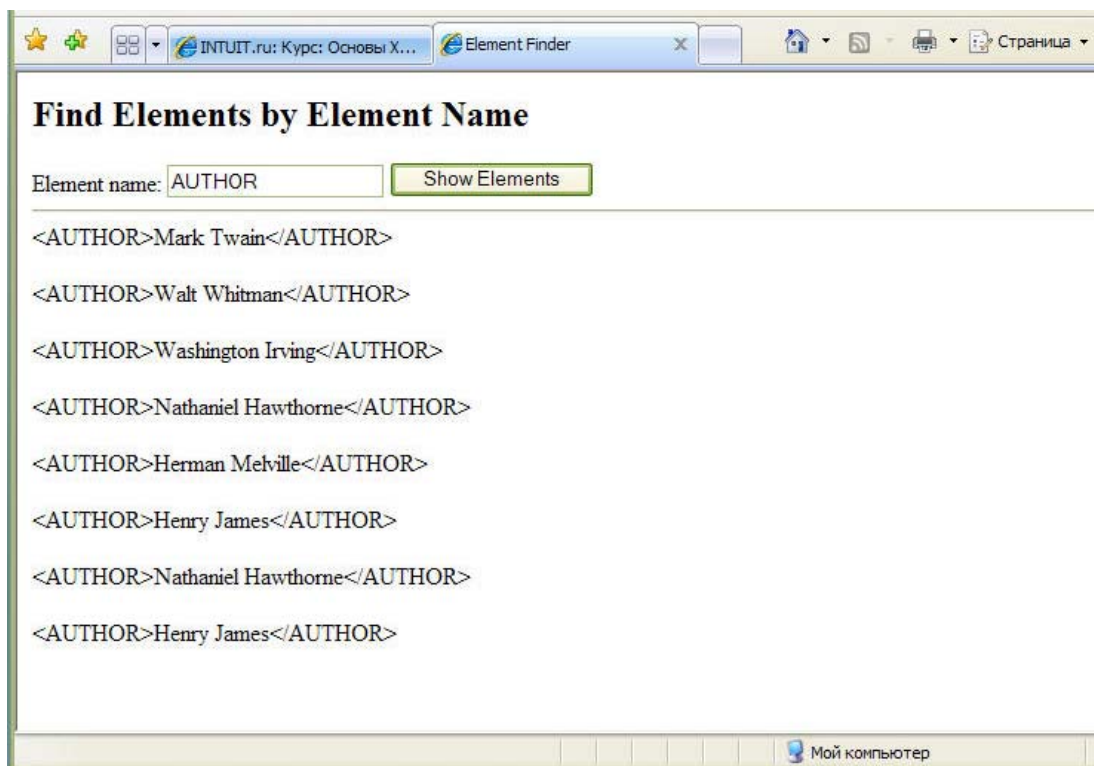


Рис. 9.4. Отображение в Internet Explorer 5 результата использования метода `getElementsByTagName`

Листинг 9.5. `GetElements.htm`

```
!-- Имя файла: GetElements.htm -->
<HTML>
<HEAD>
  <TITLE>Element Finder</TITLE>
  <SCRIPT LANGUAGE="JavaScript">
```

```

function ShowElements()
{
/* проверка ввода пользователем
                                названия элемента в поле
'Element name': */
if (ElementName.value == "")
{
    ResultDiv.innerText = "<You must enter an
                                element"
        + "name into 'Element name' box.>";
    return;
}
/* создание списка элементов NodeList
                                с именами, аналогичными
введенному: */
Document = dsoXML.XMLDocument;
NodeList =
    Document.getElementsByTagName
        (ElementName.value);
/* запись XML-указателя для каждого
найденного элемента в ResultHTML: */
ResultHTML = "";
for (i=0; i < NodeList.length; ++i)
    ResultHTML += NodeList(i).xml + "\n\n";
/* назначение сохраненных результатов в свойстве
innerText элемента DIV: */
if (ResultHTML == "")
    ResultDiv.innerText = "<no matching elements
                                found>";
else
    ResultDiv.innerText = ResultHTML;
}
</SCRIPT>
</HEAD>
<BODY>
    <XML ID="dsoXML" SRC="Inventory.xml"></XML>
    <H2>Find Elements by Element Name</H2>
    Element name: <INPUT TYPE="TEXT" ID="ElementName">&nbsp;
    <BUTTON ONCLICK="ShowElements()">Show Elements</BUTTON>
    <HR>
    <DIV ID=ResultDiv></DIV>
</BODY>
</HTML>

```

9.8. ДОСТУП И ОТОБРАЖЕНИЕ ЗНАЧЕНИЙ АТТРИБУТОВ В XML-ДОКУМЕНТЕ

Атрибут, который содержится в XML-элементе, представляется дочерним узлом Attribute. Однако вы не сможете обратиться к дочернему узлу Attribute с использованием свойств `childNodes`, `firstChild` или `lastChild`, которые годятся для доступа к дочерним узлам других типов. Вместо этого вам потребуется воспользоваться свойством `attributes` узла `Element`.

Примечание. DOM использует узлы Attribute для представления не только атрибутов, но и нескольких типов других компонентов XML, которые состоят из пар «имя – значение», а именно:

- имя и значение в инструкции по обработке (например, `version=«1.0»` в XML-объявлении);
- ключевое слово SYSTEM, за которым следует системный литерал в объявлении типа документа, объявлении внешнего примитива либо в объявлении нотации;
- ключевое слово NDATA, за которым следует имя нотации в объявлении неразбираемого примитива.

Возьмем в качестве примера XML-документ из листинга 9.6.

Листинг 9.6. Inventory Attributes.xml

```
<?xml version="1.0"?>
<!-- Имя файла: Inventory Attributes.xml -->
<INVENTORY>
  <BOOK Binding="mass market paperback" InStock="yes"
          Review="****">
    <TITLE>The Adventures of Huckleberry Finn</TITLE>
    <AUTHOR Born="1835">Mark Twain</AUTHOR>
    <PAGES>298</PAGES>
    <PRICE>$5.49</PRICE>
  </BOOK>
  <BOOK Binding="hardcover" InStock="no">
    <TITLE>Leaves of Grass</TITLE>
    <AUTHOR Born="1819">Walt Whitman</AUTHOR>
    <PAGES>462</PAGES>
    <PRICE>$7.75</PRICE>
  </BOOK>
  <BOOK Binding="mass market paperback" InStock="yes"
          Review="*****">
    <TITLE>The Legend of Sleepy Hollow</TITLE>
    <AUTHOR>Washington Irving</AUTHOR>
```

```

    <PAGES>98</PAGES>
    <PRICE>$2.95</PRICE>
  </BOOK>
</INVENTORY>

```

Элементы BOOK в этом документе имеют от двух до трех атрибутов. Следующее выражение в сценарии получает узел для первого элемента BOOK:

```
Document.documentElement.childNodes(0)
```

Свойство attributes данного узла Element предоставляет набор NamedNodeMap узлов Attribute для всех атрибутов, принадлежащих первому элементу BOOK:

```
NamedNodeMap = Document.documentElement.childNodes(0).attributes
```

Групповой объект NamedNodeMap несколько отличается от группового объекта NodeList, предоставляемого свойством узла childNodes. В табл. 9.7 приведены свойство и несколько полезных методов, поддерживаемых объектами NamedNodeMap.

Таблица 9.7

Свойство и методы, предоставляемые групповым объектом NamedNodeMap

Свойство	Описание	Пример
length	Количество узлов, содержащихся в наборе	AttributeCount = Element.attributes.length;
Метод	Описание	Пример
getNamedItem (имя-атрибута)	Возвращает узел, который носит заданное имя	Attribute = Element.attributes.getNamedItem ("Binding");
item (индекс, отсчитываемый от нуля) (метод по умолчанию)	Возвращает узел в заданной индексом позиции (нуль соответствует первому узлу)	SecondAttribute = Element.attributes.item(1); или SecondAttribute = Element.attributes(1);
reset()	Устанавливает внутренний указатель на позицию перед первым узлом в наборе, так что последующий вызов nextNode возвращает первый узел	Element.attributes.reset();
nextNode()	Возвращает следующий узел в наборе в соответствии со значением внутреннего указателя	Element.attributes.reset(); FirstAttribute = Element.attributes.nextNode();

Вы можете воспользоваться свойством `length` объекта `NamedNodeMap` и установленным по умолчанию методом `item`, чтобы перемещаться внутри набора и извлекать отдельные узлы `Attribute`. Например, следующий фрагмент сценария отображает имя и значение каждого атрибута для первого элемента `BOOK` рассматриваемого документа:

```
NamedNodeMap = Document.documentElement.childNodes(0).
                                                    attributes;
for (i=0; i<NamedNodeMap.length; ++i)
    alert ("node name: " + NamedNodeMap(i).nodeName + "\n"
        + "node value: " + NamedNodeMap(i).nodeValue);
```

Каждая пара «имя – значение» отображается в окне сообщения-предупреждения.

Обратите внимание, что свойство `nodeName` узла `Attribute` содержит имя атрибута, в то время как свойство `nodeValue` содержит значение атрибута.

⇒ *Примечание.* В действительности узел `Attribute` имеет дочерний узел `Text`, который содержит значение атрибута. Однако этот узел практически не нужен, поскольку вы легко можете получить значение атрибута из свойства `nodeValue` узла `Attribute`.

Вы можете извлечь определенный узел `Attribute` из объекта `NamedNodeMap`, вызвав метод `getNamedItem` данного объекта. Например, следующий фрагмент кода сценария отображает значение атрибута `Binding` первого элемента `BOOK` в рассматриваемом документе:

```
NamedNodeMap = Document.documentElement. childNodes(0).
                                                    attributes;
alert (NamedNodeMap.getNamedItem("Binding").nodeValue);
```

9.9. ДОСТУП К ПРИМИТИВАМ И НОТАЦИЯМ XML

Вы используете объявление неразбираемого примитива для включения в XML-документ внешних данных. (Все неразбираемые примитивы являются общими внешними примитивами.) При использовании неразбираемого примитива вы назначаете его имя

атрибуту типа ENTITY или ENTITIES, что ассоциирует файл внешнего примитива с определенным XML-элементом. XML-процессор не имеет доступа к файлу неразбираемого примитива. Он просто делает описание примитива и его нотацию доступными приложению, которое может получать и соответствующим образом использовать информацию.

В этой подглаве вы познакомитесь с XML-документом и HTML-страницей, которые демонстрируют основные этапы в использовании DOM для извлечения из XML-документа информации о примитиве, а также нотации, описывающей формат примитива. Листинг 9.7 содержит пример XML-документа, а листинг 9.8 – пример HTML-страницы.

Листинг 9.7. Inventory Entity.xml

```
<?xml version="1.0"?>
<!-- Имя файла: Inventory Entity.xml -->
<!DOCTYPE INVENTORY
[
  <!NOTATION TXT SYSTEM "plain text file">
  <!ENTITY rev_huck SYSTEM "Review of Huckleberry Finn.txt"
                                NDATA TXT>
  <!ENTITY rev_leaves SYSTEM "Review of Leaves of Grass.txt"
                                NDATA TXT>
  <!ENTITY rev_legend SYSTEM "Review of Sleepy
Hollow.txt"
                                NDATA TXT>
  <!ELEMENT INVENTORY (BOOK)*>
  <!ELEMENT BOOK (TITLE, AUTHOR, BINDING, PAGES, PRICE)>
  <!ATTLIST BOOK Review ENTITY #IMPLIED>
  <!ELEMENT TITLE (#PCDATA)>
  <!ELEMENT AUTHOR (#PCDATA)>
  <!ELEMENT BINDING (#PCDATA)>
  <!ELEMENT PAGES (#PCDATA)>
  <!ELEMENT PRICE (#PCDATA)>
]
>
<INVENTORY>
  <BOOK Review="rev_huck">
    <TITLE>The Adventures of Huckleberry Finn</TITLE>
    <AUTHOR>Mark Twain</AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>298</PAGES>
    <PRICE>$5.49</PRICE>
  </BOOK>
```



```

<BOOK Review="rev_leaves">
  <TITLE>Leaves of Grass</TITLE>
  <AUTHOR>Walt Whitman</AUTHOR>
  <BINDING>hardcover</BINDING>
  <PAGES>462</PAGES>
  <PRICE>$7.75</PRICE>
</BOOK>
<BOOK Review="rev_legend">
  <TITLE>The Legend of Sleepy Hollow</TITLE>
  <AUTHOR>Washington Irving</AUTHOR>
  <BINDING>mass market paperback</BINDING>
  <PAGES>98</PAGES>
  <PRICE>$2.95</PRICE>
</BOOK>
</INVENTORY>

```

Листинг 9.8. Inventory Entity.htm

```

<!-- Имя файла: Inventory Entity.htm -->
<HTML>
<HEAD>
  <TITLE>Get Entity Information</TITLE>
  <SCRIPT LANGUAGE="JavaScript" FOR="window" EVENT="ONLOAD">
    Document = dsoInventory.XMLDocument;
    Attribute =
      Document.documentElement.childNodes(0).attributes(0);
    if (Attribute.dataType == "entity")
    {
      DisplayText = "" + Attribute.nodeName
        + " attribute has ENTITY type" + "\n";
      DisplayText += "attribute value = "
        + Attribute.nodeValue + "\n";
      Entity = Document.doctype.entities.getNamedItem
        (Attribute.nodeValue);
      DisplayText += "entity file = "
        + Entity.attributes.getNamedItem("SYSTEM").
          nodeValue + "\n";

      NotationName =
        Entity.attributes.getNamedItem("NDATA").nodeValue;
      DisplayText += "entity notation = "
        + NotationName + "\n";
      Notation = Document.doctype.notations.getNamedItem
        (NotationName);
      DisplayText += "notation URI or description = "
        + Notation.attributes.getNamedItem("SYSTEM").
          nodeValue + "\n";
    }
  </SCRIPT>

```

```

        alert (DisplayText);
        location.href =
            Entity.attributes.getNamedItem("SYSTEM").nodeValue;
    }
</SCRIPT>
</HEAD>
<BODY>
    <XML ID="dsoInventory" SRC="Inventory Entity.xml"></XML>
</BODY>
</HTML>

```

Каждый элемент BOOK в рассматриваемом примере XML-документа содержит атрибут типа ENTITY с именем Review, которому присваивается имя неразбираемого примитива, содержащего обзор для данной книги. Пример HTML-страницы включает сценарий, демонстрирующий основные действия, которые сценарий DOM должен выполнить, чтобы извлечь всю информацию о примитиве при обнаружении атрибута с типом ENTITY или ENTITIES. В частности, сценарий извлекает информацию о неразбираемом примитиве, назначенном атрибуту Review для первого элемента BOOK. Он отображает эту информацию в окне предупреждающего сообщения (рис. 9.5).

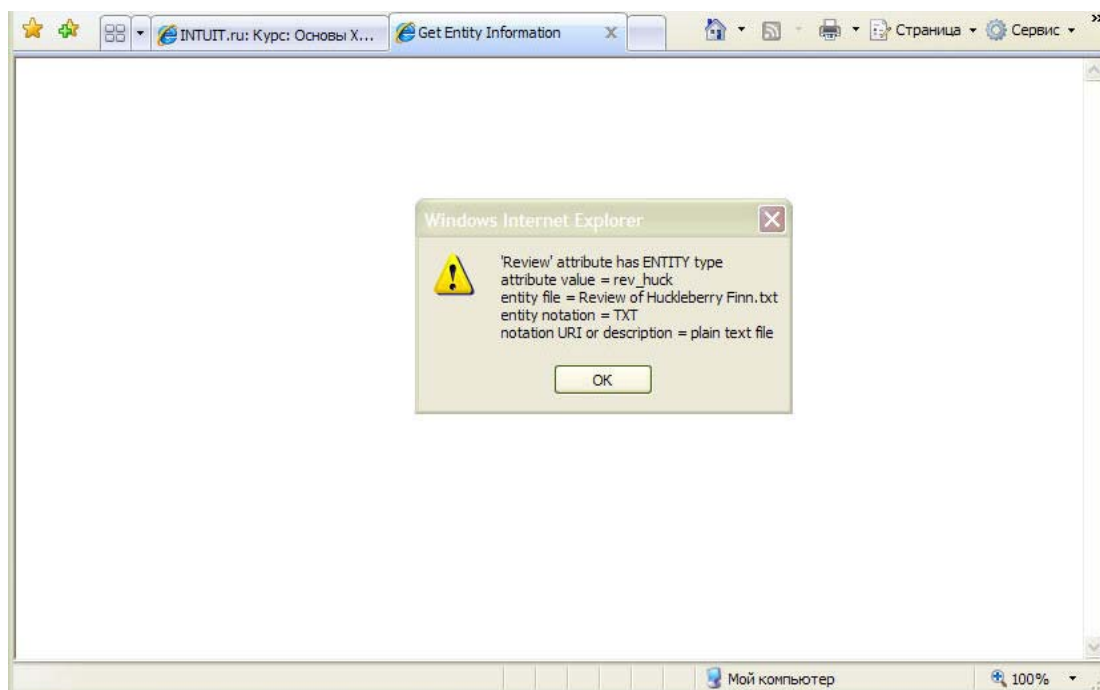


Рис. 9.5. Отображение в Internet Explorer 5 основных действий, выполняемых сценарием DOM

Ниже приведено краткое пояснение основных действий, выполняемых сценарием:

1) сценарий получает узел `Attribute` для атрибута `Review` первого элемента `BOOK`:

```
Attribute = Document.documentElement.childNodes(0).attributes(0);
```

2) сценарий использует свойство `dataType` узла (см. табл. 9.2 на с. 259–260), чтобы определить, имеет ли атрибут тип `ENTITY`:

```
if (Attribute.dataType == "entity")
```

3) сценарий выполняет остальные действия только в том случае, если атрибут имеет тип `ENTITY`, т. е. эти действия, входящие в состав оператора `if`, выполняются только тогда, когда условие `if` истинно;

4) сценарий получает узел `ENTITY` для DTD-объявления примитива, присвоенного атрибуту:

```
Entity = Document.doctype.entities.getNamedItem  
        (Attribute.nodeValue);
```

Свойство `doctype` объекта `Document` (см. табл. 9.3 на с. 265–266) предоставляет узел `DocumentType`, который есть объявление типа документа. Свойство `entities` узла `DocumentType` предоставляет набор `NamedNodeMap` узлов `Entity` для всех объявлений примитивов в DTD. Узел `Entity` для конкретного примитива, присваиваемого атрибуту, получают заданием имени примитива (`Attribute.nodeValue`) в качестве параметра метода `getNamedItem` объекта `NamedNodeMap` (см. табл. 9.7 на с. 278);

5) сценарий получает системный литерал примитива, который задает URI файла, содержащего данные примитива. Системный литерал хранится как значение узла `Attribute` с именем `SYSTEM`:

```
DisplayText += "entity file = "  
    + Entity.attributes.getNamedItem("SYSTEM").nodeValue + "\n";
```

6) сценарий получает имя нотации примитива, которое хранится как значение узла `Attribute` с именем `NDATA`:

```
NotationName = Entity.attributes.getNamedItem("NDATA").  
                nodeValue;
```

7) сценарий получает узел `Notation` для объявления нотации примитива:

```
Notation = Document.doctype.notations.getNamedItem  
        (NotationName);
```

Свойство `notations` узла `DocumentType` предоставляет набор `NamedNodeMap` узлов `Notation` для всех объявлений нотаций в DTD. Узел `Notation` для нотации примитива получают путем задания имени нотации (`NotationName`) в качестве параметра метода `getNamedItem` объекта `NamedNodeMap`;

8) сценарий получает системный литерал нотации, который содержит URI нотации или – в данном примере – ее описание. Системный литерал хранится как значение узла `Attribute` с именем `SYSTEM`:

```
DisplayText += "notation URI or description = "  
    + Notation.attributes.getNamedItem("SYSTEM").nodeValue + "\n";
```

9) сценарий отображает все хранимые результаты в окне предупреждающего сообщения:

```
alert (DisplayText);
```

10) сценарий завершается предоставлением Internet Explorer 5 возможности открыть и отобразить файл примитива, содержащего обзор. Он делает это путем присвоения URI файла свойству `location.href` HTML-страницы, которое задает URL файла, отображаемого в данный момент браузером:

```
location.href =  
Entity.attributes.getNamedItem("SYSTEM").nodeValue;
```

9.10. ПЕРЕМЕЩЕНИЕ ВНУТРИ XML-ДОКУМЕНТА

В следующем упражнении вы создадите HTML-страницу, которая содержит сценарий для перемещения внутри XML-документа среди узлов в иерархии DOM, начиная с корневого элемента `Document`. Для каждого узла сценарий отображает имя узла, тип и значение. Сценарий задает отступ для каждого блока информации в узле, чтобы показать его уровень в иерархии. Вы можете использовать эту страницу, чтобы отобразить узлы для любого XML-документа и лучше узнать, как DOM структурирует узлы для различных типов XML-документов и компонентов документа.

Создайте страницу для перемещения между узлами. Откройте новый, пустой текстовый файл в вашем текстовом редакторе и введите HTML-страницу, представленную в листинге 9.9.

Воспользуйтесь командой Save (Сохранить) вашего текстового редактора, чтобы сохранить документ на вашем жестком диске, присвоив ему имя файла ShowNodes.htm.

Листинг 9.9. ShowNodes.htm

```
<!-- Имя файла: ShowNodes.htm -->
<HTML>
<HEAD>
  <TITLE>Show DOM Nodes</TITLE>
  <SCRIPT      LANGUAGE="JavaScript"      FOR="window"
EVENT="ONLOAD">
    /* получение узла Document: */
    Document = dsoXML.XMLDocument;
    /* передача узла Document функции: */
    DisplayDIV.innerText = DisplayNodes(Document, 0);
    function DisplayNodes (Node, IndentLevel)
    {
      /* объявление локальных переменных для рекурсии: */
      var i;
      var DisplayString = "";
      /* создание отступа для данного уровня: */
      Indent = "";
      IndentDelta = "      ";
      for (i=0; i < IndentLevel; ++i)
        Indent += IndentDelta;
      /* отображение свойств текущего узла: */
      DisplayString += Indent + "nodeName: "
        + Node.nodeName + "\n"
        + Indent + "nodeType: "
        + Node.nodeType + "\n"
        + Indent + "nodeTypeString: "
        + Node.nodeTypeString + "\n"
        + Indent + "nodeValue: "
        + Node.nodeValue + "\n\n";
      /* отображение атрибутов для каждого из
                                                дочерних узлов: */
      Indent += IndentDelta;
      for (i=0;
        Node.attributes != null
        && i < Node.attributes.length;
```

```

        ++i)
    DisplayString += Indent + "nodeName: "
        + Node.attributes(i).nodeName + "\n"
        + Indent + "nodeTypeType:  "
        + Node.attributes(i).nodeType + "\n"
        + Indent + "nodeTypeString: "
        + Node.attributes(i).nodeTypeString
        + "\n"
        + Indent + "nodeValue: "
        + Node.attributes(i).nodeValue
        + "\n\n";
    /* отображение каждого из дочерних узлов,
        не являющихся атрибутами: */
    for (i=0; i < Node.childNodes.length; ++i)
        DisplayString += DisplayNodes (Node.childNodes(i),
                                        IndentLevel + 1);
    /* возврат строки, содержащей результаты: */
    return DisplayString;
}
</SCRIPT>
</HEAD>
<BODY>
    <XML ID="dsoXML" SRC="Inventory Dom.xml"></XML>
    <H2>XML Document Object Model (DOM) Nodes</H2>
    <DIV ID="DisplayDIV"></DIV>
</BODY>
</HTML>

```

В начале сценарий передает узел Document функции DisplayNodes, которая возвращает отображаемую информацию данного узла и всех его дочерних узлов. Сценарий присваивает отображаемую информацию свойству innerText элемента DisplayDIV DIV в разделе BODY страницы, который затем отображает эту информацию:

```
DisplayDIV.innerText = DisplayNodes(Document, 0);
```

Второй параметр функции DisplayNodes задает уровень отступа, используемого при отображении информации узла.

Функция DisplayNodes имеет следующую форму записи:

```
function DisplayNodes (Node, IndentLevel)
```

Функция выполняет следующие основные действия:

– сохраняет соответствующее количество символов пробелов в переменной Indent, которая используется для создания отступа

в начале каждой строки текста узла. Количество символов пробелов определяется значением параметра `IndentLevel`, передаваемого функции `DisplayNodes`:

```
/* создание отступа для данного уровня: */
Indent = "";
IndentDelta = "  ";
for (i=0; i < IndentLevel; ++i);
Indent += IndentDelta;
```

– сохраняет отображаемую информацию для текущего узла, т. е. узла, передаваемого функции `DisplayNodes` через параметр `Node` (изначально, узел `Document`):

```
/* отображение свойств текущего узла: */
DisplayString += Indent + "nodeName: "
    + Node.nodeName + "\n"
    + Indent + "nodeType: "
    + Node.nodeType + "\n"
    + Indent + "nodeTypeString: "
    + Node.nodeTypeString + "\n"
    + Indent + "nodeValue: "
    + Node.nodeValue + "\n\n";
```



Совет. Если вы хотите увидеть дополнительные свойства для каждого узла, то можете добавить их в представленный выше фрагмент кода. Вы можете применять любые общие свойства узла, приведенные в табл. 9.2 (см. на с. 259–260). Однако не следует использовать специальные свойства, характерные для определенного узла, поскольку они применимы не для всех типов узлов.

– сохраняет отображаемую информацию для дочерних узлов `Attribute` текущего узла. Отступ при этом увеличивается на один уровень, указывая на то, что эти узлы являются дочерними для текущего узла:

```
/* отображение атрибутов для каждого из дочерних узлов: */
Indent += IndentDelta;
for (i=0;
Node.attributes != null
&& i < Node.attributes.length;
++i)
DisplayString += Indent + "nodeName: "
    + Node.attributes(i).nodeName + "\n"
```

```

+ Indent + "nodeTypeType: "
+ Node.attributes(i).nodeType + "\n"
+ Indent + "nodeTypeString:"
+ Node.attributes(i).nodeTypeString
+ "\n"
+ Indent + "nodeValue: "
+ Node.attributes(i).nodeValue
+ "\n\n";

```

⇒ *Примечание.* Функция DisplayNodes не отображает дочерний узел Text узла Attribute, поскольку гораздо удобнее получить значение атрибута непосредственно из свойства nodeValue самого узла Attribute.

Функция DisplayNodes сохраняет отображаемую информацию для каждого дочернего узла, не являющегося атрибутом, осуществляя самостоятельный вызов для каждого из этих узлов. Такой вызов называется рекурсивным:

```

/* отображение каждого из дочерних узлов, не являющихся
атрибутами: */
for (i=0; i < Node.childNodes.length; ++i)
DisplayString +=
DisplayNodes (Node.childNodes(i), IndentLevel + 1);

```

Функция DisplayNodes завершает свою работу, возвращая строку, содержащую всю информацию об узле:

```

/* возврат строки, содержащей результаты: */
return DisplayString;

```

9.11. РАБОТА СТРАНИЦЫ ПРОВЕРКИ НА ВАЛИДНОСТЬ

Для HTML-страниц, с которыми вы работали в этой главе, были приняты два допущения:

1) связанный XML-документ не имеет ошибок. Если это не так, XML-данные не будут доступны;

2) браузер закончил загрузку и обработку XML-документа к тому времени, когда сценарий предпринимает попытку обратиться к данным. Если это предположение не соответствует действительности, часть данных XML будет недоступна.


```

    }
    </SCRIPT>
</HEAD>
<BODY>
<!-- Set SRC to the URL of the XML document you want
to check: -->
<XML ID="dsoTest" SRC="Inventory.xml"></XML>
</BODY>
</HTML>

```

HTML-страница содержит сценарий, который выполняется, когда браузер первый раз открывает окно страницы:

```

<SCRIPT LANGUAGE="JavaScript" FOR="window" EVENT="ONLOAD">
/* код сценария... */
</SCRIPT>

```

Сначала сценарий получает узел Document:

```
Document = dsoTest.XMLDocument;
```

Затем он проверяет свойство readyState узла Document. Если значение свойства readyState равно 4, что указывает на то, что все данные XML загружены и обработаны, сценарий сразу же вызывает функцию DisplayError, которая отображает состояние документа на предмет ошибок. Если же значение свойства readyState не равно 4, сценарий присваивает функцию DisplayError свойству onreadystatechange узла Document, что приводит к вызову браузером функции DisplayError позднее, когда значение readyState изменится:

```

if (Document.readyState == 4)
    DisplayError ();
else
    Document.onreadystatechange = DisplayError;

```

Оба этих свойства узла Document рассмотрены в табл. 9.3 (см. на с. 265–266).

Если свойство readyState еще не приняло значение 4, то функция DisplayError немедленно завершает свою работу. Если функция продолжает работу, она отображает все свойства элементарного объекта parseError узла Document. Эти свойства полностью описывают состояние XML-документа на предмет ошибок:

```

function DisplayError ()
{
    if (Document.readyState != 4)

```

```

    return;
    message = "parseError.errorCode: "
    + Document.parseError.errorCode + "\n"
    + "parseError.filepos: "
    + Document.parseError.filepos + "\n"
    + "parseError.line: " + Document.parseError.line
    + "\n"
    + "parseError.linepos: "
    + Document.parseError.linepos + "\n"
    + "parseError.reason: "
    + Document.parseError.reason + "\n"
    + "parseError.srcText: "
    + Document.parseError.srcText + "\n"
    + "parseError.url: " + Document.parseError.url;
    alert (message);
}

```

Если документ не содержит ошибок, значение `parseError.errorCode` устанавливается в нуль, а другие свойства также имеют нулевые или пустые значения. Если же в документе есть ошибка, свойство `parseError.errorCode` содержит числовой код ошибки, а другие свойства описывают эту ошибку.

Проверка валидности XML-документа

На рис. 9.6 приведена HTML-страница, которая открывает XML-документ и использует свойства DOM для выдачи сообщений относительно обнаруженных в нем ошибок. Если документ не имеет объявления типа документа, страница выдает сообщения только об ошибках корректности формы. Если же документ включает объявление типа документа, страница выдает сообщения как об ошибках корректности формы, так и об ошибках валидности. Вы можете использовать эту страницу для тестирования любого XML-документа.

ЗАДАНИЯ

1. Откройте страницу из листинга 9.1 в Internet Explorer 5

Обратите внимание, что свойство `nodeTypeString` содержит тип узла в виде строчных букв. (Так, «Document» и «ProcessingInstruction» превращаются в «document» и «processinginstruction».)

Изначально страница отображает XML-документ `Inventory Dom.xml` (рис. 9.6).

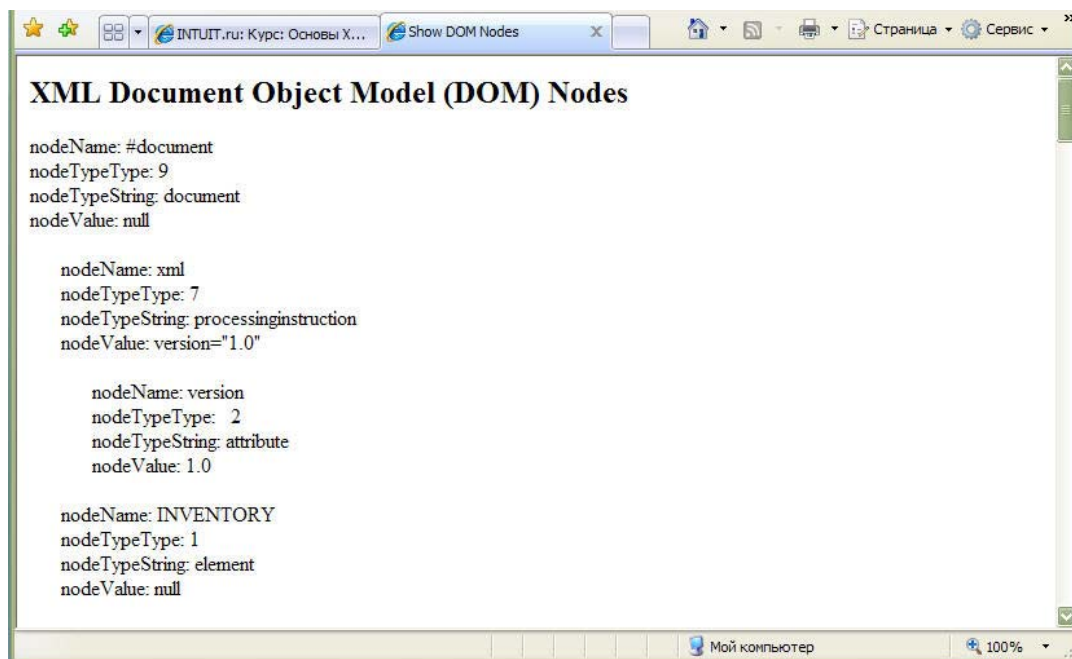


Рис. 9.6. Отображение в Internet Explorer 5 файла Inventory Dom.xml

Чтобы просмотреть структуру узлов для других XML-документов, отредактируйте фрагмент данных страницы. Например, чтобы просмотреть узлы документа Inventory Valid Entity.xml, вы должны изменить фрагмент данных следующим образом:

```
<XML ID="dsoXML" SRC="Inventory Valid Entity.xml"></XML>
```

2. Как использовать страницу проверки на валидность

В вашем текстовом редакторе откройте страницу проверки на валидность Validity Test.htm. Отредактируйте фрагмент данных в разделе BODY страницы, чтобы атрибуту SRC было присвоено URL XML-документа, который вы хотите подвергнуть тестированию. Например, чтобы протестировать документ Raven.xml, вы должны отредактировать фрагмент данных следующим образом:

```
<XML ID="dsoTest" SRC="Raven.xml"></XML>
```

Воспользуйтесь командой Save (Сохранить) вашего текстового редактора, чтобы сохранить модифицированную страницу.

Откройте страницу в Internet Explorer 5.

Страница отобразит окно сообщения, содержащее информацию о первой ошибке, обнаруженной XML-процессором Internet Explorer 5. На рис. 9.7 показано, как будет выглядеть окно сообщения, если документ не содержит ошибок.

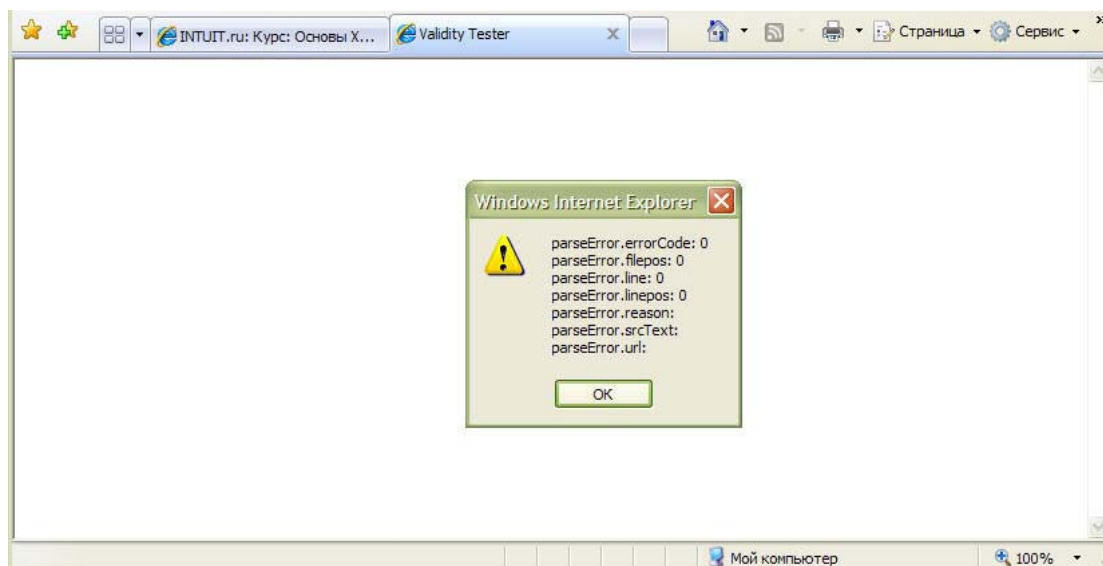


Рис. 9.7. Отображение в Internet Explorer 5 окна сообщения об отсутствии ошибок в документе

А на рис. 9.8 показано, как будет выглядеть окно сообщения при наличии ошибки в документе.

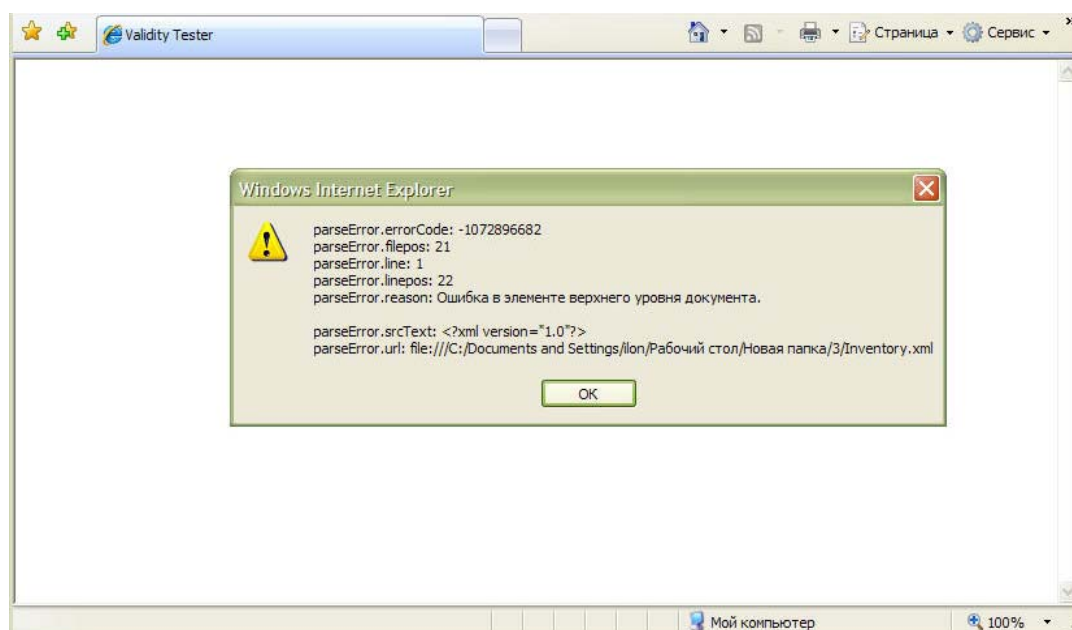


Рис. 9.8. Отображение в Internet Explorer 5 окна сообщения о наличии ошибок в документе

Страница выдает сообщения об ошибках корректности формы. Также выдается сообщение об ошибках валидности документа, так как был объявлен его тип.

Глава 10. ФИЛЬТРАЦИЯ И СОРТИРОВКА ДАННЫХ XML

10.1. ОСНОВЫ ИСПОЛЬЗОВАНИЯ XSL-ТАБЛИЦ СТИЛЕЙ

Существуют два основных шага для отображения XML-документа при использовании XSL-таблицы стилей:

1) создание файла XSL-таблицы стилей. XSL является приложением XML, иными словами, XSL-таблица представляет собой корректно сформированный XML-документ, который отвечает правилам XSL. Подобно любому XML-документу, XSL-таблица стилей содержит простой текст, и вы можете создать ее с помощью вашего любимого текстового редактора. В последующих подглавах рассказывается, как создавать различные типы XSL-таблиц стилей;

2) связывание XSL-таблицы стилей с XML-документом. Вы можете связать XSL-таблицу стилей с XML-документом, включив в документ инструкцию по обработке `xml-stylesheet`, которая имеет следующую обобщенную форму записи:

```
<?xml-stylesheet type="text/xsl" href=XSLFilePath?>
```

Здесь `XSLFilePath` представляет собой заключенный в кавычки URL, указывающий местонахождение файла таблицы стилей. Вы можете использовать полный URL, например:

```
<?xml-stylesheet type="text/xsl"
href="http://www.my_domain.com/Inventory.xsl"?>
```

Чаще применяют неполный URL, который задает местонахождение относительно месторасположения XML-документа, содержащего инструкцию по обработке `xml-stylesheet`, например:

```
<?xml-stylesheet type="text/xsl" href="Inventory.xsl"?>
```

(Относительный URL встречается чаще, поскольку вы обычно храните файл таблицы стилей в той же папке, где хранится XML-документ, либо в одной из вложенных в нее папок.)



Совет. Хотя вы и можете связать XSL-таблицу стилей с использованием полного URL, таблица стилей при этом должна размещаться на том же домене, что и XML-документ, с которым вы ее связываете. Например, если домен <http://mspress.microsoft.com/> содержит XML-документ, то и XSL-таблица стилей должна размещаться на том же домене.

Обычно инструкция по обработке `xml-styleSheet` добавляется в пролог XML-документа вслед за объявлением XML, как вы увидите в примере XML-документа, рассматриваемого в следующей подглаве (см. листинг 10.2 на с. 296–297). Более подробная информация об инструкциях по обработке и описании мест в документе, куда они могут быть корректно помещены, содержится в 4.3 «Добавление инструкций по обработке».

Если вы связали XSL-таблицу стилей с XML-документом, вы можете открыть этот документ непосредственно в Internet Explorer 5, и браузер отобразит XML-документ с использованием инструкций по преобразованию, содержащихся в таблице стилей. В отличие от таблиц каскадных стилей, если вы связываете с XML-документом более одной XSL-таблицы стилей, браузер использует первую таблицу и игнорирует все остальные. Если вы свяжете с XML-документом и CSS-таблицу, и XSL-таблицу стилей, браузер использует только XSL-таблицу стилей.



Примечание. Если вы не связали XML-документ ни с CSS-таблицей, ни с XSL-таблицей стилей, Internet Explorer 5 отобразит документ с помощью встроенной XSL-таблицы, которая используется по умолчанию. Эта таблица стилей отображает исходный XML-текст в виде дерева с возможностью свертывания/развертывания уровней.

10.2. ИСПОЛЬЗОВАНИЕ ОДНОГО ШАБЛОНА XSL

В отличие от CSS, содержащей правила, XSL-таблица стилей включает один или несколько шаблонов, каждый из которых содержит информацию для отображения в определенной ветви элементов в XML-документе. В этой подглаве вы узнаете, как создать простую XSL-таблицу стилей, которая включает только один шаблон. Этот шаблон содержит информацию для отображения всего документа.

В листинге 10.1 представлен первый пример XSL-таблицы стилей. Эта таблица стилей связана с XML-документом, представленным в листинге 10.2.

Листинг 10.1. XslDemo01.xsl

```
<?xml version="1.0"?>
<!-- Имя файла: XslDemo01.xsl -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template match="/">
    <H2>Book Description</H2>
    <SPAN STYLE="font-style:italic">Author: </SPAN>
    <xsl:value-of select="BOOK/AUTHOR"/><BR/>
    <SPAN STYLE="font-style:italic">Title: </SPAN>
    <xsl:value-of select="BOOK/TITLE"/><BR/>
    <SPAN STYLE="font-style:italic">Price: </SPAN>
    <xsl:value-of select="BOOK/PRICE"/><BR/>
    <SPAN STYLE="font-style:italic">Binding type: </SPAN>
    <xsl:value-of select="BOOK/BINDING"/><BR/>
    <SPAN STYLE="font-style:italic"> Number of pages: </SPAN>
    <xsl:value-of select="BOOK/PAGES"/>
  </xsl:template>
</xsl:stylesheet>
<?xml version="1.0"?>
<!-- Имя файла: XslDemo01.xml -->
<?xml-stylesheet type="text/xsl" href="XslDemo01.xsl"?>
<BOOK>
  <TITLE>Moby-Dick</TITLE>
  <AUTHOR>
    <FIRSTNAME>Herman</FIRSTNAME>
    <LASTNAME>Melville</LASTNAME>
  </AUTHOR>
  <BINDING>hardcover</BINDING>
  <PAGES>724</PAGES>
  <PRICE>$9.95</PRICE>
</BOOK>
```

Листинг 10.2. XslDemo01.xml

```
<?xml version="1.0"?>
<!-- Имя файла: XslDemo01.xml -->
<?xml-stylesheet type="text/xsl" href="XslDemo01.xsl"?>
<BOOK>
  <TITLE>Moby-Dick</TITLE>
  <AUTHOR>
    <FIRSTNAME>Herman</FIRSTNAME>
    <LASTNAME>Melville</LASTNAME>
  </AUTHOR>
```



```

<BINDING>hardcover</BINDING>
<PAGES>724</PAGES>
<PRICE>$9.95</PRICE>
</BOOK>

```

На рис. 10.1 показано, как будет выглядеть документ в браузере.

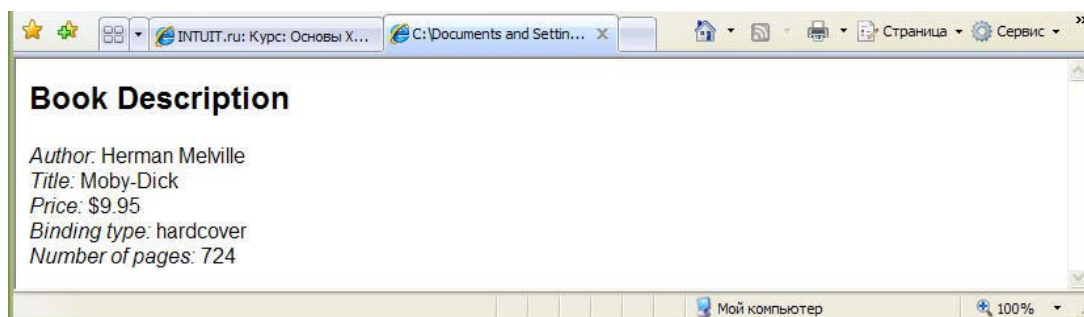


Рис. 10.1. Отображение в Internet Explorer 5 XSL-таблицы стилей

Каждая XSL-таблица стилей должна иметь элемент Документ, представленный ниже. (Напомним, что элемент Документ, известный как корневой элемент, является XML-элементом верхнего уровня, который содержит все остальные элементы.)

```

<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <!-- Один или несколько элементов шаблона... -->
</xsl:stylesheet>

```

Элемент Документ `xsl:stylesheet` служит не только хранилищем других элементов, но также идентифицирует документ как XSL-таблицу стилей. Этот элемент является одним из XSL-элементов специального назначения, используемых в таблице стилей. Все XSL-элементы принадлежат пространству имен `xsl`, т. е. вы предваряете имя каждого XSL-элемента префиксом `xsl:`, обозначающим пространство имен. Вы определяете это пространство имен в начальном теге элемента `xsl:stylesheet`, например, следующим образом:

```
xmlns:xsl="http://www.w3.org/TR/WD-xsl"
```

Это определение позволяет вам использовать пространство имен внутри элементов таблицы стилей. (Относительно пространства имен в XML смотрите в 7.13 «Вставка элементов HTML в XML-документы и использование пространства имен»).

Элемент Документ `xsl:stylesheet` XSL-таблицы стилей должен содержать один или несколько шаблонов элементов, которые для

краткости будем называть шаблонами. Элемент Документ из листинга 10.1 содержит только один шаблон, который имеет следующую форму:

```
<xsl:template match="/">
  <!-- Дочерние элементы.. -->
</xsl:template>
```

Браузер использует шаблон для отображения определенной ветви элементов в иерархии XML-документа, с которым вы связываете таблицу стилей. Атрибут `match` шаблона указывает на определенную ветвь. (Атрибут `match` аналогичен селектору в правиле CSS.) Значение атрибута `match` носит название образца (pattern). Образец в данном примере ("/") представляет корневой элемент всего XML-документа. Этот шаблон, таким образом, содержит инструкции для отображения всего XML-документа.

Каждая XSL-таблица стилей должна содержать один и только один шаблон с атрибутом `match`, который имеет значение "/". Вы также можете включить один или несколько дополнительных шаблонов с инструкциями для отображения определенных подчиненных ветвей в структуре XML-документа; каждая из них должна иметь образец, отвечающий определенной ветви.

Корневой образец ("/") не представляет элемент Документ (или корневой элемент) XML-документа, а представляет весь документ, для которого элемент Документ является дочерним, т. е. он аналогичен корневому узлу Document в объектной модели документа DOM.

Ниже приведено полное описание шаблона из рассматриваемой таблицы стилей:

```
<xsl:template match="/">
  <H2>Book Description</H2>
  <SPAN STYLE="font-style:italic">Author: </SPAN>
  <xsl:value-of select="BOOK/AUTHOR"/><BR/>
  <SPAN STYLE="font-style:italic">Title: </SPAN>
  <xsl:value-of select="BOOK/TITLE"/><BR/>
  <SPAN STYLE="font-style:italic">Price: </SPAN>
  <xsl:value-of select="BOOK/PRICE"/><BR/>
  <SPAN STYLE="font-style:italic">Binding type: </SPAN>
  <xsl:value-of select="BOOK/BINDING"/><BR/>
  <SPAN STYLE="font-style:italic">Number of pages: </SPAN>
  <xsl:value-of select="BOOK/PAGES"/>
</xsl:template>
```

Шаблон содержит два вида XML-элементов:
– XML-элементы, представляющие HTML-разметку. Примерами подобного вида XML-элемента из рассматриваемой таблицы стилей являются:

```
<H2>Book Description</H2>
```

который отображает заголовок второго уровня,

```
<SPAN STYLE="font-style:italic">Author: </SPAN>
```

который отображает блок текста, набранного курсивом (Author:),

```
<BR/>
```

который создает пустую строку.

Все эти XML-элементы являются корректно сформированными и представляют стандартные HTML-элементы. Браузер просто копирует каждый HTML-элемент непосредственно на выход HTML, который воспринимает и отображает их.



Совет. Каждый из элементов, представляющих HTML-разметку, должен быть корректно сформированным XML-элементом, а также стандартным HTML-элементом. (Не забывайте, что XSL-таблица стилей является XML-документом.) Следовательно, вы не можете использовать HTML-конструкции, которые не являются корректно сформированным XML, такие как элементы, состоящие только из начального тега. Например, чтобы задать элемент перевода строки в HTML, вы не можете просто ввести `
`, как вы это делаете для HTML-страницы. Вместо этого вы должны использовать корректно сформированный тег пустого XML-элемента, `
`.

– XSL-элементы. Примеры XSL-элементов из рассматриваемой таблицы стилей являются элементами `xsl:value-of`:

```
<xsl:value-of select="BOOK/AUTHOR"/>
```

Браузер отличает XML-элемент от элемента, представляющего HTML, поскольку первый имеет в качестве префикса описание пространства имен `xsl:`. XSL-элементы в шаблоне не копируются на выход HTML. Они лишь содержат инструкции по выбору и модификации данных XML, либо используются для выполнения других задач.

XSL-элемент `value-of` добавляет текстовое содержимое определенного XML-элемента, а также любых его дочерних элементов, которые он имеет, в выходной модуль HTML, воспринимаемый и отображаемый

браузером. Вы указываете определенный XML-элемент заданием образца, который присваиваете атрибуту `select` XSL-элемента `value-of`. В рассмотренном выше примере элемента `value-of` атрибуту `select` присвоен образец «BOOK/AUTHOR», что приводит к выводу текстового содержимого элемента `AUTHOR` XML-документа. Текстовое содержимое элемента `AUTHOR` состоит из символьных данных, принадлежащих двум его дочерним элементам (`FIRSTNAME` и `LASTNAME`).

Обратите внимание, что XML-элемент в образце задается с помощью оператора пути (в данном случае `BOOK/AUTHOR`), который определяет местонахождение элемента в иерархии XML-документа. (Оператор пути похож на путь к файлу, который операционная система использует для указания местонахождения файла или папки.)

Главный момент, на который здесь следует обратить внимание, состоит в том, что оператор пути в значении атрибута `select` относится к текущему элементу. Каждый контекст внутри XSL-таблицы стилей принадлежит к текущему элементу. Поскольку рассматриваемый пример шаблона относится к корневому элементу всего документа (посредством установки атрибута `match="/"`), текущим элементом для данного шаблона является корневой элемент документа. (В данном случае текущий элемент не обладает соответствующим литералом, а является родителем элемента Документ.) Таким образом, внутри этого шаблона оператор пути `BOOK/AUTHOR` указывает на элемент `AUTHOR`, вложенный в элемент `BOOK`, который относится к корневому элементу документа.

Если вы опустите атрибут `select` для XSL-элемента `value-of`, элемент будет осуществлять вывод текстового содержимого плюс текстового содержимого всех дочерних элементов в текущий элемент. (В нашем примере, поскольку текущим является корневой элемент, пропуск атрибута `select` приведет к выводу всех символьных данных в XML-документ.)

Целью представленного в рассматриваемом примере шаблона элементов является отображение текста названия для каждого из дочерних XML-элементов в документе (`AUTHOR`, `TITLE`, `PRICE`, `BINDING` и `PAGES`) плюс текстового содержимого каждого элемента. Обратите внимание, что порядок элементов `value-of` в шаблоне определяет порядок, в котором браузер отображает эти элементы. Таким образом, даже из этой простой таблицы стилей вы можете понять, что XSL-таблица стилей является гораздо более гибкой, чем CSS, которая всегда отображает элементы в том порядке, в котором они следуют в документе.

Итак, как вы могли заметить, XSL-таблица стилей сообщает браузеру, как отобразить XML-документ путем избирательного преобразования XML-элементов в блок HTML-разметки, который воспринимается и отображается браузером аналогично разметке, содержащейся на HTML-странице. Подчеркнем, однако, что вам не нужно включать в XSL-шаблон элементы, представляющие элементы HTML или BODY, которые являются стандартными составными частями HTML-страницы, поскольку браузер сам эффективно их формирует.

На рис. 10.2 показано, как браузер генерирует первую часть блока HTML-разметки для документа и таблицы стилей из листингов 10.1 и 10.2.

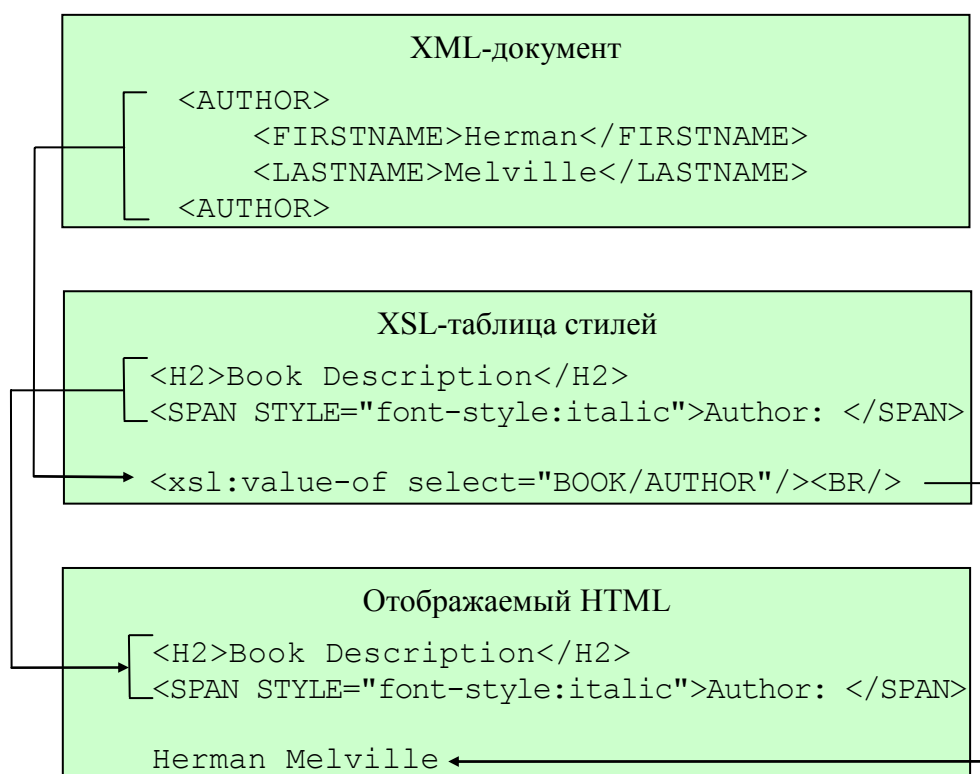


Рис. 10.2. Генерация браузером первой части блока HTML-разметки

10.3. ОТОБРАЖЕНИЕ ПЕРЕМЕННОГО ЧИСЛА ЭЛЕМЕНТОВ

В примере, рассмотренном в предыдущей подглаве (см. листинг 10.2 на с. 296–297), XML-документ содержал только один элемент BOOK. В случае если документ содержит несколько элементов

BOOK, методика, с которой вы познакомились в предыдущей подглаве, способна отобразить только один из элементов. Возьмем, для примера, XML-документ, содержащий следующий элемент Документ:

```
<INVENTORY>
  <BOOK>
    <TITLE>The Adventures of Huckleberry Finn</TITLE>
    <AUTHOR>
      <FIRSTNAME>Mark</FIRSTNAME>
      <LASTNAME>Twain</LASTNAME>
    </AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>298</PAGES>
    <PRICE>$5.49</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>The Adventures of Tom Sawyer</TITLE>
    <AUTHOR>
      <FIRSTNAME>Mark</FIRSTNAME>
      <LASTNAME>Twain</LASTNAME>
    </AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>205</PAGES>
    <PRICE>$4.75</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>The Ambassadors</TITLE>
    <AUTHOR>
      <FIRSTNAME>Henry</FIRSTNAME>
      <LASTNAME>James</LASTNAME>
    </AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>305</PAGES>
  </BOOK>
</INVENTORY>
```

Предположим, что таблица стилей, используемая для отображения этого документа, содержит следующий шаблон:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template match="/">
    <H2>Book Description</H2>
    <SPAN STYLE="font-style:italic">Author: </SPAN>
    <xsl:value-of select="INVENTORY/BOOK/AUTHOR"/><BR/>
    <SPAN STYLE="font-style:italic">Title: </SPAN>
    <xsl:value-of select="INVENTORY/BOOK/TITLE"/><BR/>
    <SPAN STYLE="font-style:italic">Price: </SPAN>
```

```

<xsl:value-of select="INVENTORY/BOOK/PRICE"/><BR/>
<SPAN STYLE="font-style:italic">Binding type: </SPAN>
<xsl:value-of select="INVENTORY/BOOK/BINDING"/><BR/>
<SPAN STYLE="font-style:italic">Number of pages:</SPAN>
<xsl:value-of select="INVENTORY/BOOK/PAGES"/>
</xsl:template>
</xsl:stylesheet>

```

Этот шаблон использует методику, описанную в предыдущей подглаве. Обратите внимание, что образец присваиваемых каждому атрибуту `select` начинается с указания элемента Документ, в данном случае `INVENTORY` (например, «`INVENTORY/BOOK/AUTHOR`»).

Каждый образец, однако, соответствует трем различным элементам. Например, «`INVENTORY/BOOK/AUTHOR`» соответствует элементу `AUTHOR` для всех трех элементов `BOOK`. В подобной ситуации браузер использует только первый из соответствующих элементов. Таким образом, таблица стилей отобразит содержимое только первого элемента `BOOK`, как показано на рис. 10.3.

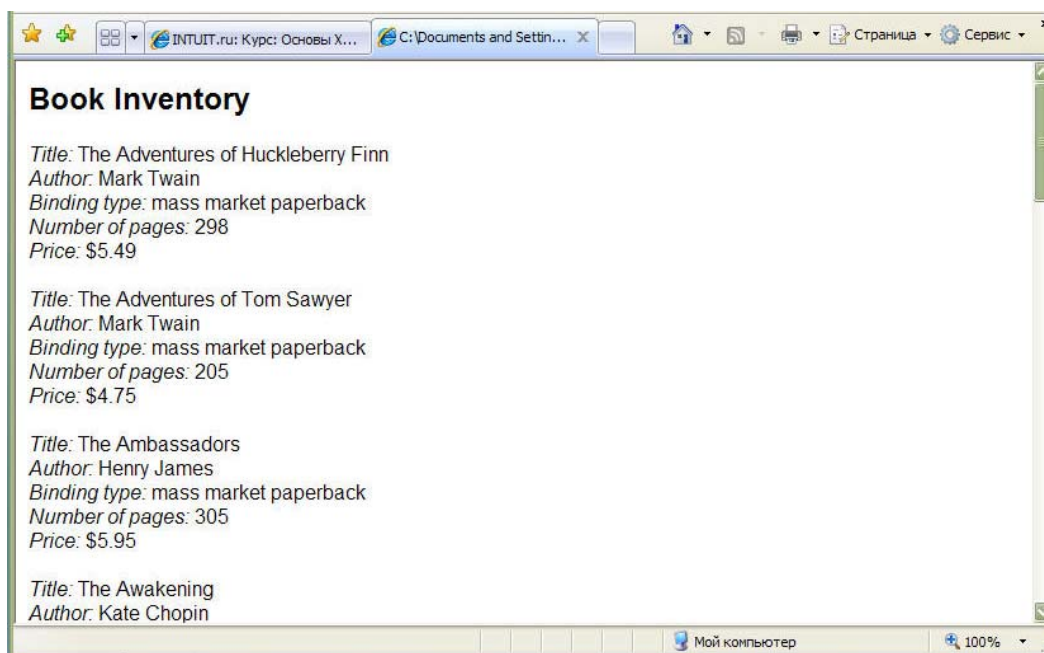


Рис. 10.3. Отображение в Internet Explorer 5 содержимого только первого элемента `BOOK`

Чтобы отобразить все отвечающие образцу элементы, следует использовать XSL-элемент `for-each`, который вызывает повторный вывод для каждого из содержащихся в XML-файле элементов.

XSL-таблица стилей, представленная в листинге 10.3, демонстрирует данную методику. Эта таблица стилей связана с XML-документом, содержащимся в листинге 10.4.

Листинг 10.3. XslDemo02.xsl

```
<?xml version="1.0"?>
<!-- Имя файла: XslDemo02.xsl -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template match="/">
    <H2>Book Inventory</H2>
    <xsl:for-each select="INVENTORY/BOOK">
      <SPAN STYLE="font-style:italic">Title: </SPAN>
      <xsl:value-of select="TITLE"/><BR/>
      <SPAN STYLE="font-style:italic">Author: </SPAN>
      <xsl:value-of select="AUTHOR"/><BR/>
      <SPAN STYLE="font-style:italic">Binding type: </SPAN>
      <xsl:value-of select="BINDING"/><BR/>
      <SPAN STYLE="font-style:italic">Number of pages: </SPAN>
      <xsl:value-of select="PAGES"/><BR/>
      <SPAN STYLE="font-style:italic">Price: </SPAN>
      <xsl:value-of select="PRICE"/><P/>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

Листинг 10.4. XslDemo.xml

```
<?xml version="1.0"?>
<!-- Имя файла: XslDemo.xml -->
<?xml-stylesheet type="text/xsl" href="XslDemo02.xsl"?>
<INVENTORY>
  <BOOK>
    <TITLE>The Adventures of Huckleberry Finn</TITLE>
    <AUTHOR>
      <FIRSTNAME>Mark</FIRSTNAME>
      <LASTNAME>Twain</LASTNAME>
    </AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>298</PAGES>
    <PRICE>$5.49</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>The Adventures of Tom Sawyer</TITLE>
    <AUTHOR>
      <FIRSTNAME>Mark</FIRSTNAME>
      <LASTNAME>Twain</LASTNAME>
    </AUTHOR>
```



```

        <BINDING>mass market paperback</BINDING>
        <PAGES>205</PAGES>
        <PRICE>$4.75</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>The Ambassadors</TITLE>
        <AUTHOR>
            <FIRSTNAME>Henry</FIRSTNAME>
            <LASTNAME>James</LASTNAME>
        </AUTHOR>
        <BINDING>mass market paperback</BINDING>
        <PAGES>305</PAGES>
        <PRICE>$5.95</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>The Awakening</TITLE>
        <AUTHOR>
            <FIRSTNAME>Kate</FIRSTNAME>
            <LASTNAME>Chopin</LASTNAME>
        </AUTHOR>
        <BINDING>mass market paperback</BINDING>
        <PAGES>195</PAGES>
        <PRICE>$4.95</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>Billy Budd</TITLE>
        <AUTHOR>
            <FIRSTNAME>Herman</FIRSTNAME>
            <LASTNAME>Melville</LASTNAME>
        </AUTHOR>
        <BINDING>mass market paperback</BINDING>
        <PAGES>195</PAGES>
        <PRICE>$4.49</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>A Connecticut Yankee in King Arthur's
                                   Court</TITLE>
        <AUTHOR>
            <FIRSTNAME>Mark</FIRSTNAME>
            <LASTNAME>Twain</LASTNAME>
        </AUTHOR>
        <BINDING>mass market paperback</BINDING>
        <PAGES>385</PAGES>
        <PRICE>$5.49</PRICE>
    </BOOK>
    <BOOK>

```

```

<TITLE>Joan of Arc</TITLE>
<AUTHOR>
  <FIRSTNAME>Mark</FIRSTNAME>
  <LASTNAME>Twain</LASTNAME>
</AUTHOR>
<BINDING>trade paperback</BINDING>
<PAGES>465</PAGES>
<PRICE>$6.95</PRICE>
</BOOK>
<BOOK>
  <TITLE>Leaves of Grass</TITLE>
  <AUTHOR>
    <FIRSTNAME>Walt</FIRSTNAME>
    <LASTNAME>Whitman</LASTNAME>
  </AUTHOR>
  <BINDING>hardcover</BINDING>
  <PAGES>462</PAGES>
  <PRICE>$7.75</PRICE>
</BOOK>
<BOOK>
  <TITLE>The Legend of Sleepy Hollow</TITLE>
  <AUTHOR>
    <FIRSTNAME>Washington</FIRSTNAME>
    <LASTNAME>Irving</LASTNAME>
  </AUTHOR>
  <BINDING>mass market paperback</BINDING>
  <PAGES>98</PAGES>
  <PRICE>$2.95</PRICE>
</BOOK>
<BOOK>
  <TITLE>The Marble Faun</TITLE>
  <AUTHOR>
    <FIRSTNAME>Nathaniel</FIRSTNAME>
    <LASTNAME>Hawthorne</LASTNAME>
  </AUTHOR>
  <BINDING>trade paperback</BINDING>
  <PAGES>473</PAGES>
  <PRICE>$10.95</PRICE>
</BOOK>
<BOOK>
  <TITLE>Moby-Dick</TITLE>
  <AUTHOR>
    <FIRSTNAME>Herman</FIRSTNAME>
    <LASTNAME>Melville</LASTNAME>
  </AUTHOR>
  <BINDING>hardcover</BINDING>

```

```

        <PAGES>724</PAGES>
        <PRICE>$9.95</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>Passing</TITLE>
        <AUTHOR>
            <FIRSTNAME>Nella</FIRSTNAME>
            <LASTNAME>Larsen</LASTNAME>
        </AUTHOR>
        <BINDING>trade paperback</BINDING>
        <PAGES>165</PAGES>
        <PRICE>$5.95</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>The Portrait of a Lady</TITLE>
        <AUTHOR>
            <FIRSTNAME>Henry</FIRSTNAME>
            <LASTNAME>James</LASTNAME>
        </AUTHOR>
        <BINDING>mass market paperback</BINDING>
        <PAGES>256</PAGES>
        <PRICE>$4.95</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>Roughing It</TITLE>
        <AUTHOR>
            <FIRSTNAME>Mark</FIRSTNAME>
            <LASTNAME>Twain</LASTNAME>
        </AUTHOR>
        <BINDING>mass market paperback</BINDING>
        <PAGES>324</PAGES>
        <PRICE>$5.25</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>The Scarlet Letter</TITLE>
        <AUTHOR>
            <FIRSTNAME>Nathaniel</FIRSTNAME>
            <LASTNAME>Hawthorne</LASTNAME>
        </AUTHOR>
        <BINDING>trade paperback</BINDING>
        <PAGES>253</PAGES>
        <PRICE>$4.25</PRICE>
    </BOOK>
    <BOOK>
        <TITLE>The Turn of the Screw</TITLE>
        <AUTHOR>

```

```

        <FIRSTNAME>Henry</FIRSTNAME>
        <LASTNAME>James</LASTNAME>
    </AUTHOR>
    <BINDING>trade paperback</BINDING>
    <PAGES>384</PAGES>
    <PRICE>$3.35</PRICE>
</BOOK>
</INVENTORY>

```

Шаблон в таблице стилей из листинга 10.3 содержит следующий элемент `for-each`:

```

<xsl:for-each select="INVENTORY/BOOK">
    <SPAN STYLE="font-style:italic">Title: </SPAN>
    <xsl:value-of select="TITLE"/><BR/>
    <SPAN STYLE="font-style:italic">Author: </SPAN>
    <xsl:value-of select="AUTHOR"/><BR/>
    <SPAN STYLE="font-style:italic">Binding type: </SPAN>
    <xsl:value-of select="BINDING"/><BR/>
    <SPAN STYLE="font-style:italic">Number of pages: </SPAN>
    <xsl:value-of select="PAGES"/><BR/>
    <SPAN STYLE="font-style:italic">Price: </SPAN>
    <xsl:value-of select="PRICE"/><P/>
</xsl:for-each>

```

Элемент `for-each` выполняет две основные задачи:

1) осуществляет вывод блока элементов, содержащихся внутри элемента `for-each`, повторяя его для каждого XML-элемента в документе, отвечающего образцу, который присвоен атрибуту `select` элемента `for-each`. В данном примере цикл выполняется по одному разу для каждого элемента `BOOK`, найденного в элементе Документ с именем `INVENTORY`. Образец, присваиваемый атрибуту `select`, работает точно так же, как образец, присваиваемый атрибуту `select` элемента `value-of`;

2) задает текущий элемент, устанавливаемый атрибутом `select` элемента `for-each` (`/INVENTORY/BOOK` в нашем примере указывает на элемент `BOOK` внутри элемента `INVENTORY`, входящего в корневой элемент документа) следующим образом:

```

<xsl:stylesheet xmlns:xsl=http://www.w3.org/TR/WD-xsl>
    <xsl:template match="/">
        <!-- Здесь текущим является корневой элемент
                                     документа, "/" . -->

```

```

    <xsl:for-each select="INVENTORY/BOOK">
    <!-- Здесь текущим является элемент /INVENTORY/BOOK. -->
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>

```

Аналогично внутри элемента `for-each` каждый дочерний элемент может быть выбран путем задания образца, содержащего только имя элемента, например:

```

<xsl:value-of select="TITLE"/>

```

10.4. ИСПОЛЬЗОВАНИЕ НЕСКОЛЬКИХ ШАБЛОНОВ

Другой способ отображения повторяющихся XML-элементов состоит в создании отдельного шаблона для каждого элемента с последующим вызовом этого шаблона с использованием XSL-элемента `apply-templates`. Пример использования подобной методики приведен в XSL-таблице стилей, представленной в листинге 10.5. Эта таблица стилей предназначена для связывания с XML-документом из листинга 10.4. Установить эту связь вы можете путем модификации инструкции `xml-stylesheet` в документе следующим образом.

Листинг 10.5. XslDemo03.xml

```

<?xml version="1.0"?>
<!-- Имя файла: XslDemo03.xml -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template match="/">
    <H2>Book Inventory</H2>
    <xsl:apply-templates select="INVENTORY/BOOK" />
  </xsl:template>
  <xsl:template match="BOOK">
    <SPAN STYLE="font-style:italic">Title: </SPAN>
    <xsl:value-of select="TITLE"/><BR/>
    <SPAN STYLE="font-style:italic">Author: </SPAN>
    <xsl:value-of select="AUTHOR"/><BR/>
    <SPAN STYLE="font-style:italic">Binding type: </SPAN>
    <xsl:value-of select="BINDING"/><BR/>
  </xsl:template>
</xsl:stylesheet>

```

```

        <SPAN STYLE="font-style:italic">Number of pages: </SPAN>
        <xsl:value-of select="PAGES"/><BR/>
        <SPAN STYLE="font-style:italic">Price: </SPAN>
        <xsl:value-of select="PRICE"/><P/>
    </xsl:template>
</xsl:stylesheet>

```

Рассматриваемая в примере таблица стилей состоит из двух шаблонов. Один шаблон содержит инструкции для отображения всего документа (путем установки `match="/"`, указывающей на корневую часть документа). Все XSL-таблицы стилей требуют наличия такого шаблона. Другой шаблон содержит инструкции для отображения элемента BOOK. Сначала браузер обрабатывает шаблон, соответствующий корневой части элемента:

```

<xsl:template match="/">
    <H2>Book Inventory</H2>
    <xsl:apply-templates select="INVENTORY/BOOK"/>
</xsl:template>

```

XSL-элемент `apply-templates` сообщает браузеру, что для каждого элемента BOOK внутри корневого элемента INVENTORY он должен обрабатывать шаблон, отвечающий элементу BOOK, т. е. шаблон, для атрибута `match` которого установлено значение «BOOK». Таблица стилей включает следующий шаблон, отвечающий элементу BOOK:

```

<xsl:template match="BOOK">
    <SPAN STYLE="font-style:italic">Title: </SPAN>
    <xsl:value-of select="TITLE"/><BR/>
    <SPAN STYLE="font-style:italic">Author: </SPAN>
    <xsl:value-of select="AUTHOR"/><BR/>
    <SPAN STYLE="font-style:italic">Binding type: </SPAN>
    <xsl:value-of select="BINDING"/><BR/>
    <SPAN STYLE="font-style:italic">Number of pages: </SPAN>
    <xsl:value-of select="PAGES"/><BR/>
    <SPAN STYLE="font-style:italic">Price: </SPAN>
    <xsl:value-of select="PRICE"/><P/>
</xsl:template>

```

Браузер обрабатывает шаблон BOOK один раз для каждого элемента BOOK, отображая всю информацию о книгах, имеющуюся в документе (рис. 10.4).

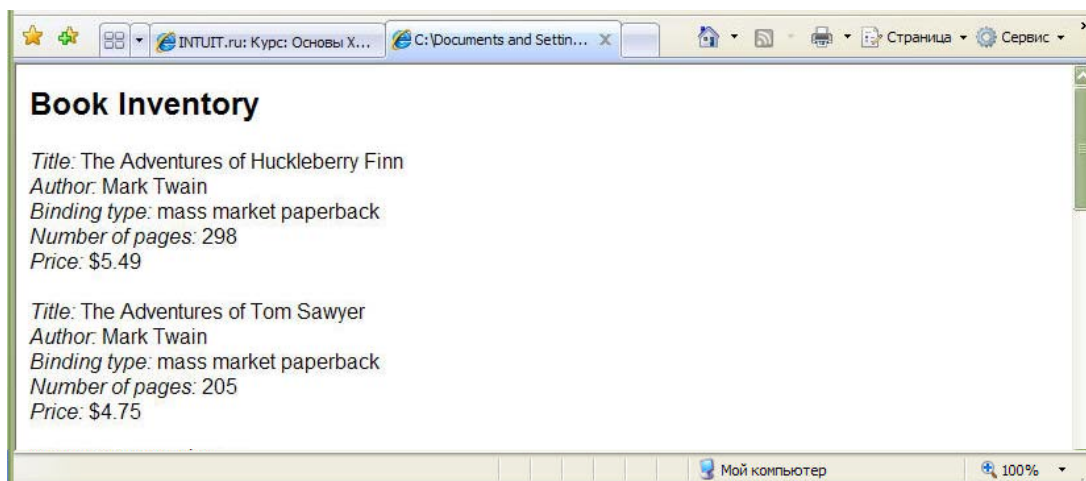


Рис. 10.4. Отображение в Internet Explorer 5 результата использования XSL-элемента `apply-templates`

Поскольку этот шаблон отвечает элементу `BOOK`, элемент `BOOK` является текущим элементом в контексте шаблона. В связи с этим доступ к дочерним элементам `BOOK` осуществляется посредством образца, содержащего только имя элемента, как в нашем примере:

```
<xsl:value-of select="TITLE"/>
```



Примечание. Если вы не укажете атрибут `select` для элемента `apply-templates`, браузер обрабатывает соответствующий шаблон (если он имеется) для каждого дочернего элемента текущего элемента. В рассматриваемом примере элемента `apply-templates` единственным дочерним элементом для текущего элемента (корневая часть документа) является элемент `INVENTORY`, который не имеет соответствующего шаблона. Таким образом, если вы опустите атрибут `select`, никакие данные не будут выведены.

10.5. ФИЛЬТРАЦИЯ И СОРТИРОВКА ДАННЫХ XML

В последующих двух пунктах вы познакомитесь с основами применения XSL-таблиц стилей для фильтрации и сортировки данных XML. После этого вам будут представлены примеры таблиц стилей, демонстрирующие методики фильтрации и сортировки.

10.5.1. Фильтрация

Значение, которое присваивается атрибутам `match` или `select`, представляет собой образец, соответствующий одному или нескольким элементам в XML-документе. (Атрибут `match` используется для элемента `template`, а атрибут `select` – для элементов `value-of`, `for-each` и `apply-templates`.) Образцы, с которыми вы имели дело до сих пор, содержали только оператор пути, который задавал имя элемента и, возможно, одного или нескольких вложенных элементов. Вы можете ограничить количество элементов, отвечающих шаблону, введя фильтр – выражение, заключенное в квадратные скобки (`[]`) и следующее непосредственно оператором пути. Например, образец, присвоенный следующему атрибуту `match`, указывает, что соответствующий элемент должен носить имя `BOOK` и, кроме того (это определяется фильтром), должен иметь дочерний элемент `BINDING`, который содержит текст «trade paperback»:

```
<xsl:template match="BOOK[BINDING='trade paperback']">
```

Если в фильтр включено только имя элемента, то соответствующий элемент должен иметь дочерний элемент с указанным именем. Например, следующий образец отвечает любому элементу `ITEM`, имеющему дочерний элемент с именем `CD`, независимо от содержимого элемента `CD`:

```
match="ITEM[CD] "
```

Нижеприведенный образец отвечает любому элементу `SHIRT`, имеющему дочерний элемент `COLOR`, который содержит текст «red»:

```
match="SHIRT [COLOR='red'] "
```

А следующий образец, наоборот, отвечает любому элементу `SHIRT`, имеющему дочерний элемент `COLOR`, который не содержит текст «red»:

```
select="SHIRT [COLOR='red'] "
```

⇒ *Примечание.* Если элемент имеет более одного дочернего элемента с именем, указанным в условии фильтрации, оператор сравнения применяется только к первому дочернему элементу. Например, если элемент `SHIRT` имеет два дочерних элемента `COLOR`, образец `"SHIRT[COLOR='red']"` будет отвечать элементу, только если первый элемент `COLOR` содержит слово «red».

10.5.2. Сортировка

В этой главе вы уже познакомились с двумя элементами, которые вы можете использовать для обработки повторяющихся элементов: `for-each` и `apply-templates`. Вы можете использовать атрибут `order-by` для этих элементов, чтобы управлять порядком, в котором браузер обрабатывает элементы, тем самым осуществляя сортировку данных XML.

Вы можете назначать атрибуту `order-by` один или несколько образцов, разделяя их точкой с запятой. Браузер будет сортировать элементы с использованием образцов в том порядке, в котором они перечислены. Для указания направления сортировки (по возрастанию или по убыванию) следует предварить образец префиксом `+` или `-`.

Например, атрибут `order-by`, установленный для следующего элемента `for-each`, предписывает браузеру сортировать элементы `BOOK` по фамилиям авторов в порядке возрастания и осуществлять сортировку для одинаковых фамилий по именам также по возрастанию:

```
<xsl:for-each select="INVENTORY/BOOK"
order-by="+AUTHOR/LASTNAME; +AUTHOR/FIRSTNAME">
```

В другом примере следующая установка `order-by` сортирует элементы `BOOK` по названиям книг в порядке убывания:

```
<xsl:apply-templates select="INVENTORY/BOOK" order-by="-TITLE">
```

Оператор пути, который вы присваиваете атрибуту `order-by`, действует относительно образца, назначенного атрибуту `select`. Так, в данном примере установка `order-by= "-TITLE"` указывает на элемент `TITLE` внутри элемента `BOOK`, вложенного в элемент `INVENTORY`.

10.5.3. Пример таблицы стилей, осуществляющей фильтрацию и сортировку

В этом пункте приведено два примера XSL-таблиц стилей, представленных в листингах 10.6 и 10.7. Каждый из них осуществляет фильтрацию и сортировку элементов `BOOK`, подлежащих отображению.

Листинг 10.6. Xsldemo04.xsl

```
<?xml version="1.0"?>
<!-- Имя файла: XslDemo04.xsl -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
```

```

<xsl:template match="/">
  <H2>Book Inventory</H2>
  <xsl:for-each
    select="INVENTORY/BOOK[BINDING='trade paperback']"
    order-by="+AUTHOR/LASTNAME; +AUTHOR/FIRSTNAME">
    <SPAN STYLE="font-style:italic">Author: </SPAN>
    <xsl:value-of select="AUTHOR"/><BR/>
    <SPAN STYLE="font-style:italic">Title: </SPAN>
    <xsl:value-of select="TITLE"/><BR/>
    <SPAN STYLE="font-style:italic">Binding type: </SPAN>
    <xsl:value-of select="BINDING"/><BR/>
    <SPAN STYLE="font-style:italic">Number of pages:
      </SPAN>
    <xsl:value-of select="PAGES"/><BR/>
    <SPAN STYLE="font-style:italic">Price: </SPAN>
    <xsl:value-of select="PRICE"/><P/>
  </xsl:for-each>
</xsl:template>
</xsl:stylesheet>

```

Листинг 10.7. Xsldemo05.xsl

```

<?xml version="1.0"?>
<!-- Имя файла: XslDemo05.xsl -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template match="/">
    <H2>Book Inventory</H2>
    <xsl:apply-templates select="INVENTORY/BOOK"
      order-by="+AUTHOR/LASTNAME; +AUTHOR/FIRSTNAME"/>
  </xsl:template>
  <xsl:template match="BOOK[BINDING='trade paperback']">
    <SPAN STYLE="font-style:italic">Author: </SPAN>
    <xsl:value-of select="AUTHOR"/><BR/>
    <SPAN STYLE="font-style:italic">Title: </SPAN>
    <xsl:value-of select="TITLE"/><BR/>
    <SPAN STYLE="font-style:italic">Binding type: </SPAN>
    <xsl:value-of select="BINDING"/><BR/>
    <SPAN STYLE="font-style:italic">Number of pages: </SPAN>
    <xsl:value-of select="PAGES"/><BR/>
    <SPAN STYLE="font-style:italic">Price: </SPAN>
    <xsl:value-of select="PRICE"/><P/>
  </xsl:template>
</xsl:stylesheet>

```

Обе таблицы стилей разработаны для связывания с XML-документом из листинга 10.4. В них использован следующий фильтр,

предписывающий браузеру отображать только книги, которые имеют прошитый бумажный переплет (trade paperback):

```
[BINDING='trade paperback']
```

В обоих примерах используется следующая установка order-by, задающая сортировку элементов BOOK по возрастанию по фамилиям авторов, а затем по именам авторов:

```
order-by="+AUTHOR/LASTNAME; +AUTHOR/FIRSTNAME"
```

На рис. 10.5 показано, как выглядит первая часть выводимой информации, которая является одинаковой для обеих таблиц стилей.

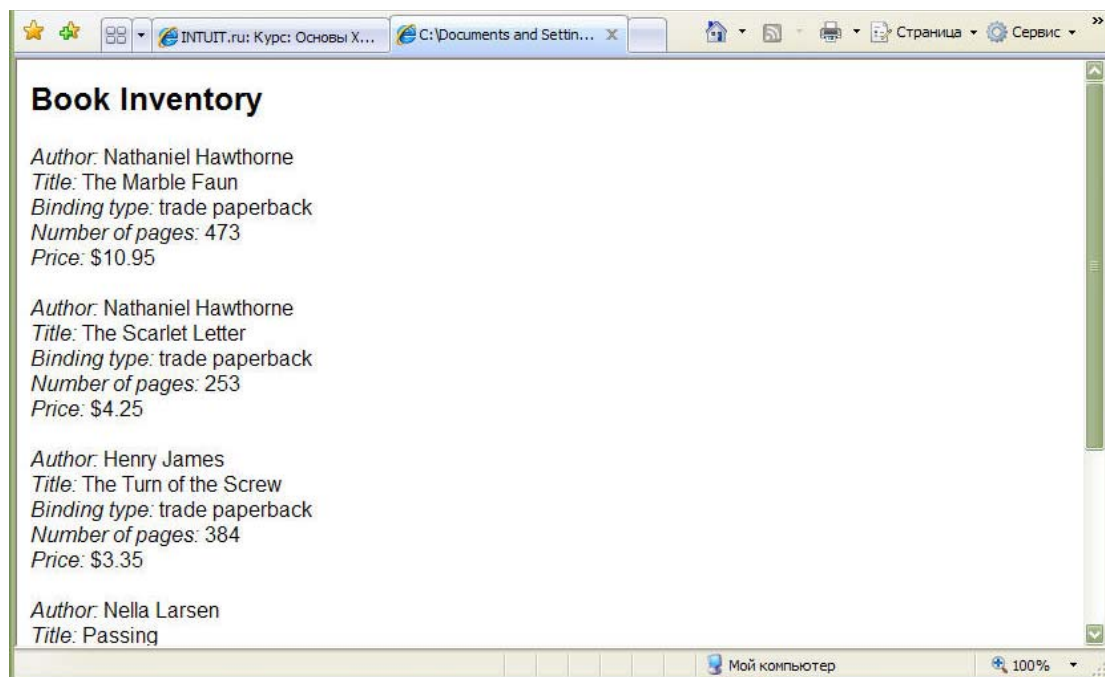


Рис. 10.5. Отображение в Internet Explorer 5 XML-документа с установкой order-by

Таблица стилей из листинга 10.6 использует элемент for-each для отображения множества элементов BOOK. В приведенной ниже таблице стилей для элемента for-each установлены и фильтр, и атрибут order-by:

```
<xsl:for-each  
  select="INVENTORY/BOOK[BINDING='trade paperback']"  
  order-by="+AUTHOR/LASTNAME; +AUTHOR/FIRSTNAME">  
  <!-- Отображение текущего элемента BOOK. -->  
</xsl:for-each>
```

Таблица стилей из листинга 10.7 использует для отображения множества элементов BOOK элемент `apply-templates` вместе с отдельными шаблонами, отвечающими элементам BOOK. В этой таблице стилей фильтр добавлен к шаблону, соответствующему элементам BOOK:

```
<xsl:template match="BOOK[BINDING='trade paperback']">
```

Добавление фильтра к элементу `apply-templates` будет иметь тот же эффект.

Атрибут `order-by` может быть добавлен к элементу `apply-templates` следующим образом:

```
<xsl:apply-templates select="INVENTORY/BOOK"
order-by="+AUTHOR/LASTNAME; +AUTHOR/FIRSTNAME"/>
```

Атрибут `order-by` следует добавить к элементу `apply-templates`, поскольку элемент `template` не распознает этот атрибут.

10.6. ДОСТУП К АТТРИБУТАМ XML

XSL трактует атрибут, принадлежащий элементу в XML-документе, как дочерний элемент. Однако для ссылки на атрибут в образце XSL вы должны предварить имя атрибута символом `@`, что указывает на то, что имя относится к атрибуту, а не к элементу.

Например, фильтр в следующем начальном теге выделяет все элементы BOOK с атрибутом `InStock`, имеющем значение «yes». Другими словами, он выбирает только книги, которые имеются в наличии:

```
<xsl:for-each select="INVENTORY/BOOK[@InStock='yes']">
```

Вы можете использовать XSL-элемент `value-of` для извлечения значений атрибута точно так же, как вы это делаете для извлечения текстового содержимого элемента. Например, следующий элемент `value-of` получает значение атрибута `Born`, принадлежащего элементу `AUTHOR`:

```
<xsl:value-of select="AUTHOR/@Born"/>
```

Таблица стилей, представленная в листинге 10.8, демонстрирует технику доступа к атрибутам, принадлежащим элементам в XML-документе. Эта таблица стилей связана с XML-документом из листинга 10.9 и отображает все имеющиеся в наличии книги из каталога.

Листинг 10.8. XslDemo06.xsl

```
<?xml version="1.0"?>
<!-- Имя файла: XslDemo06.xsl -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template match="/">
    <H2>Books In Stock</H2>
    <TABLE BORDER="1" CELLPADDING="5">
      <THEAD>
        <TH>Title</TH>
        <TH>Author</TH>
        <TH>Binding Type</TH>
        <TH>Number of Pages</TH>
        <TH>Price</TH>
      </THEAD>
      <xsl:for-each select="INVENTORY/BOOK
                          [@InStock='yes']">
        <TR ALIGN="CENTER">
          <TD>
            <xsl:value-of select="TITLE"/>
          </TD>
          <TD>
            <xsl:value-of select="AUTHOR"/><BR/>
            (born <xsl:value-of select="AUTHOR/
                          @Born"/>)
          </TD>
          <TD>
            <xsl:value-of select="BINDING"/>
          </TD>
          <TD>
            <xsl:value-of select="PAGES"/>
          </TD>
          <TD>
            <xsl:value-of select="PRICE"/>
          </TD>
        </TR>
      </xsl:for-each>
    </TABLE>
  </xsl:template>
</xsl:stylesheet>
```

Листинг 10.9. XslDemo06.xml

```
<?xml version="1.0"?>
<!-- Имя файла: XslDemo06.xml -->
<?xml-stylesheet type="text/xsl" href="XslDemo06.xsl"?>
<INVENTORY>
  <BOOK InStock="yes">
```

```

        <TITLE>The Adventures of Huckleberry Finn</TITLE>
        <AUTHOR Born="1835">Mark Twain</AUTHOR>
        <BINDING>mass market paperback</BINDING>
        <PAGES>298</PAGES>
        <PRICE>$5.49</PRICE>
    </BOOK>
    <BOOK InStock="no">
        <TITLE>Leaves of Grass</TITLE>
        <AUTHOR Born="1819">Walt Whitman</AUTHOR>
        <BINDING>hardcover</BINDING>
        <PAGES>462</PAGES>
        <PRICE>$7.75</PRICE>
    </BOOK>
    <BOOK InStock="yes">
        <TITLE>The Marble Faun</TITLE>
        <AUTHOR Born="1804">Nathaniel Hawthorne</AUTHOR>
        <BINDING>trade paperback</BINDING>
        <PAGES>473</PAGES>
        <PRICE>$10.95</PRICE>
    </BOOK>
    <BOOK InStock="yes">
        <TITLE>Moby-Dick</TITLE>
        <AUTHOR Born="1819">Herman Melville</AUTHOR>
        <BINDING>hardcover</BINDING>
        <PAGES>724</PAGES>
        <PRICE>$9.95</PRICE>
    </BOOK>
</INVENTORY>

```

Каждый элемент **BOOK** в XML-документе содержит атрибут **InStock**, имеющий значение «yes» или «no» и указывающий наличие или отсутствие книги в хранилище. Каждый элемент **AUTHOR** имеет атрибут **Born**, содержащий год рождения автора.

Вместо отображения значения атрибута **InStock** таблица стилей использует атрибут в условии фильтрации с целью избежать отображения элементов **BOOK** для книг, которых нет в наличии:

```

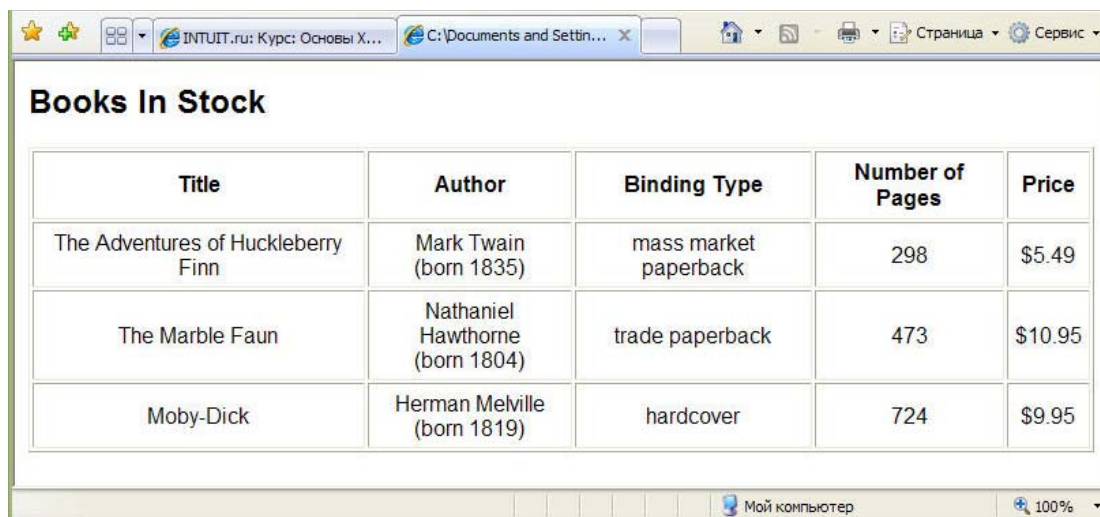
<xsl:for-each select="INVENTORY/BOOK[@InStock='yes']">
    <!-- Отображение каждого элемента BOOK. -->
</xsl:for-each>

```

Таблица стилей отображает каждый элемент **BOOK** в виде HTML-таблицы (рис. 10.6), а не через список элементов **SPAN**, как в предыдущих примерах. Она отображает значение атрибута **Born** после значения элемента **AUTHOR**, используя XSL-элемент **value-of**.

Следующие элементы создают ячейку таблицы для отображения этих значений:

```
<TD>
  <xsl:value-of select="AUTHOR"/> <BR/>
  (born <xsl:value-of select="AUTHOR/@Born"/>)
</TD>
```



The screenshot shows a web browser window with the title 'Books In Stock'. The browser's address bar shows 'INTUIT.ru: Курс: Основы X...'. The page content is an XML table with five columns: Title, Author, Binding Type, Number of Pages, and Price. The table lists three books: 'The Adventures of Huckleberry Finn' by Mark Twain, 'The Marble Faun' by Nathaniel Hawthorne, and 'Moby-Dick' by Herman Melville. The browser's status bar at the bottom shows 'Мой компьютер' and '100%' zoom.

Title	Author	Binding Type	Number of Pages	Price
The Adventures of Huckleberry Finn	Mark Twain (born 1835)	mass market paperback	298	\$5.49
The Marble Faun	Nathaniel Hawthorne (born 1804)	trade paperback	473	\$10.95
Moby-Dick	Herman Melville (born 1819)	hardcover	724	\$9.95

Рис. 10.6. Отображение в Internet Explorer 5 элементов BOOK в виде HTML-таблицы

? ЗАДАНИЯ

Создайте свою XML-страницу аналогично примерам, рассмотренным в листингах 10.5–10.9.

Используйте сортировку и фильтрацию элементов.

ЛИТЕРАТУРА

1. Чанг, Б. Oracle9i XML. Разработка приложений электронной коммерции с использованием технологии XML / Б. Чанг, М. Скардина, С. Киритцов. – М.: Лори, 2003. – 476 с.
2. Чемберлин, Д. W3C XML: Хвату от экспертов. Руководство по языку запросов / Д. Чемберлин. – М.: КУДИЦ-Образ, 2005. – 480 с.
3. XML. Работа с XML / Д. Хантер [и др.]. – 4-е изд. – Киев: Диалектика, 2009. – 1344 с.
4. Дарахвелидзе, П. Г. Разработка Web-служб средствами Delphi / П. Г. Дарахвелидзе, Е. П. Марков. – СПб.: БХВ, 2003. – 653 с.
5. Шеперд, Д. Освой самостоятельно XML за 21 день / Д. Шеперд; пер. с англ. – 2-е изд. – М.: Вильямс, 2002. – 432 с.
6. Рэй, Э. Изучаем XML / Э. Рэй. – М.: Символ-Плюс, 2001. – 408 с.
7. Гарольд, Э. Р. XML. Справочник / Э. Р. Гарольд, В. Скотт. – М.: Символ-Плюс, 2010. – 576 с.
8. XML. Базовый курс / Д. Хантер [и др.]. – М.: Вильямс, 2009. – 1344 с.
9. Тейбор, Р. Реализация XML Web-служб на платформе Microsoft. NET XML Web Services / Р. Тейбор. – М.: Вильямс, 2002. – 464 с.

Учебное издание

Кобайло Александр Серафимович
Жиляк Надежда Александровна

ВВЕДЕНИЕ В XML

Учебно-методическое пособие

Редактор *Е. С. Ватеичкина*
Компьютерная верстка *Е. В. Ильченко*
Корректор *Е. С. Ватеичкина*

Подписано в печать 16.11.2011. Формат 60×84¹/₁₆.
Бумага офсетная. Гарнитура Таймс. Печать офсетная.
Усл. печ. л. 18,7. Уч.-изд. л. 19,3.
Тираж 150 экз. Заказ .

Издатель и полиграфическое исполнение:
УО «Белорусский государственный технологический университет».
ЛИ № 02330/0549423 от 08.04.2009.
ЛП № 02330/0150477 от 16.01.2009.
Ул. Свердлова, 13а, 220006, г. Минск.

Переплетно-брошюровочные процессы произведены
в ОАО «Полиграфкомбинат им. Я. Коласа».
Ул. Красная, 23, 220600, г. Минск. Заказ .